# ExperienceReport

## WebSphere Application Server and Lotus Domino Scenario Overview

Cindy Murch
Daniel Boyum
Sue Kelling
Marie Wilson

November 2002

# Table of Contents

ExperienceReport

# WebSphere(R) Application Server and Lotus(R) Domino(TM) Scenario Overview

This report documents the iSeries(TM) System Test team's experience of bringing WebSphere Application Server and Lotus Domino together as part of a single application. In this application, we use Lotus Domino to establish an initial Web presence, create a database, and utilize directory services. WebSphere Application Server is used to create dynamic Web pages, set up single sign on, and ensure security. This was done through several phases as part of our flights scenario.

The flights scenario simulates an airline company which is working hard to increase their market share. The flights company determined that in order to remain competitive, they needed to have a Web site in which customers could view and book flights. Anxious to get a Web presence, they decided to make this transition in several phases which included:

1. Getting an initial presence by allowing the customers to view the available flights
2. Increasing customer services by allowing customers to book flights on-line
3. Branching out to create a business to business relationship with a travel agency

Each of these phases built on the previous phase and incorporated the latest technologies available. Below are more details on the functions and technologies used in each of the phases.

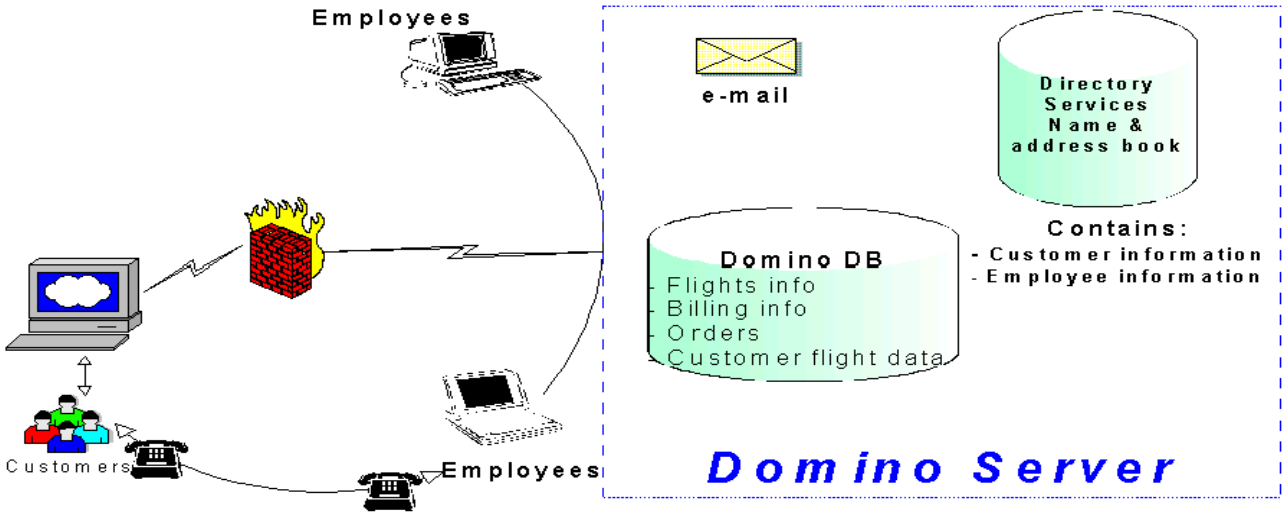
Phase one consisted of creating a web presence via static Lotus Domino served pages. During this phase a Lotus Domino Server was used for a database, directory services, and the Lotus Domino HTTP server.

Phase one allowed the customers and employees to do the following functions:

- Customers
  - View available flights from the Web site
  - Call a flights employee to book a flight
- Employees
  - Add and delete flights
  - Add, update, and delete customer information
  - Book flights
  - Retrieve customer flight information
  - Bill customers and update billing information

Figure 1 illustrates phase one.

**Figure 1 Flights scenario phase one**



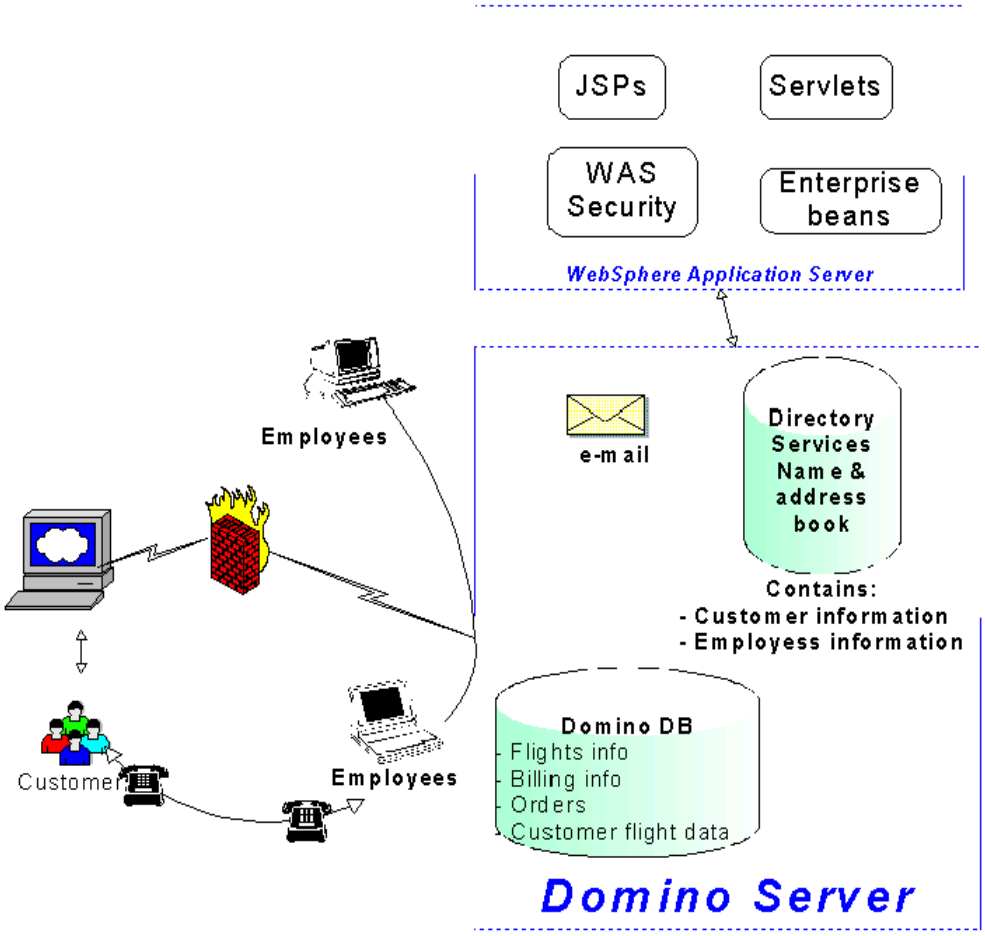

Phase two consisted of using the existing Lotus Domino server, database, and directory services. In addition, WebSphere Application Server was used to provide dynamic Web sites which allowed the customers to book flights. Servlets, enterprise beans, and JSPs were used to upgrade the scenario. WebSphere Application Server single sign-on (SSO) and security were used to complete the application.

As the result of the phase two changes, customers were able to do the functions listed above as well as these additional functions:

- Book available flights from the Web site
- Retrieve customer flight information
- Update customer information
- Pay bills on-line

Figure 2 illustrates phase two.

**Figure 2 Flights scenario phase two**




Phase three, which is yet to be implemented, will consist of creating a business to business relationship with a travel agency. This relationship will allow customers to book flights through the travel agency.

For more information on the Lotus Domino environment, the WebSphere Application Server environment, or the interoperability between the two, see the following sections.

***Lotus Domino environment***
This section describes the overall Lotus Domino environment. It includes an overview of the environment, the workflow throughout the environment, and our key findings from creating and using this environment.

***WebSphere Application Server environment***
This section describes the overall WebSphere Application Server environment. It includes an overview of the environment, the application flow through the environment, and our key findings from creating and using this environment.

***WebSphere Application Server and Lotus Domino inter operability***
This section describes the inter operability of the WebSphere Application Server and the Lotus Domino environment. It includes an overview, details about setting up the single sign-on, and our key findings from creating and using WebSphere Application Server and Lotus Domino together.

# Lotus(R) Domino(TM) environment overview

The Lotus Domino environment was set up during phase one of the iSeries(TM) System Test flights scenario work. During phase one, the goal was to quickly establish a Web presence so the flights company could expand their business and provide a solution that was reliable, available, scalable, and could integrate easily with other applications.

These requirements led the flights company to choose Lotus Domino on eServer iSeries. With Lotus Domino, the flights company was able to quickly obtain a Web presence for their customers and employees by providing static Web pages. In addition, Lotus Domino allowed the flights company to take advantage of workflow processing and provided inter operability with many other platforms.

The flights company's Lotus Domino application contains customer, flight, itinerary, and billing information. The application uses various forms and views, which create and maintain the information. The flight information consists of the flights offered including the arrival dates, departure dates, times, cities, and the number of first class and coach seats available per flight. To securely manage both the customer and employee data, a Lotus Domino Lightweight Directory Access Protocol (LDAP) directory is used.

The flights company provides two Web interfaces using Lotus Domino framesets. There is one interface for the customers and one for the employees.

The flights customers can perform the following tasks:

- View flights
- Call a flights employee to book a flight

The flights employees can perform the following tasks:

- Add, display, and delete flights
- Add and delete city codes
- Add, update, display, and delete customer information
- Book flights
- Retrieve customer flight information
- Display and update billing information

During phase two, WebSphere(R) Application Server was used to allow more customer functions including the ability to book flights. See the WebSphere Application Server Overview for more information.

# Lotus(R) Domino(TM) environment workflow

## Application Design Points

When designing the Lotus Domino application, the flights company had to choose between using framesets or navigators for their Web interface. Since the flights company wanted to provide a consistent structure throughout the website that could display different forms and views, and since frames can contain forms, folders, pages, documents and views, the frameset became the best choice.

Another decision the flights company faced was finding and implementing the best way to secure their data. They wanted a solution that was robust enough to provide the security they needed, integrated easily with their existing application, and had the flexibility to integrate with other software packages. These requirements led the flights company to choose Lotus Domino Lightweight Directory Access Protocol (LDAP).

Since the flights company already had several employees with skills in Java(TM) development, they were able to use those skills in writing the background agents for their application in Java. This proved valuable in terms of making the best use of the skills of their employees.

## Application Setup

For all of our Lotus Domino implementation work, we used Lotus Domino Designer as the development tool. Lotus Domino designer is a Lotus Notes(R) client application used to quickly create and modify a Lotus Domino application. It provides the application building blocks for everything needed in a database, including forms, views, and agents. We used forms to create new documents in a database and display current documents. Views provide a flexible and intuitive way for documents to be organized. Users can easily see lists of documents, sort the lists in different ways, open documents for reading or editing, and create new documents.

# Lotus Domino forms and views

The flights company employees created the forms and views shown in Table 1 for their Lotus Domino application.

**Table 1 Flights Lotus Domino Forms and Views**

| Lotus Domino forms | Lotus Domino views |
|---|---|
| List Of Available Flights | List Of Available Flights |
| Scheduled Flights | Processed Flights, Scheduled Flights |
| Airplane Seat Information | Airplane Seat Information |
| Ticket Information | Active Tickets, Inactive Tickets, Processed Tickets |
| Credit Information | Credit Information |
| City Codes | City Codes |
| Bill Information | Bill Information |
| Customer Number | viewCustomerFldNumber |
| Flight Number | viewFlightFldNumber |
| Invoice Number | viewInvoiceFldNumber |
| Seat Number | viewSeatFldNumber |
| Statement Number | viewStatementFldNumber |

The List Of Available Flights form contains the flight information for the available flights that are offered by the flight company. It is used to create a new flight or display an existing flight. All of the flight specific data is contained in this form. The cities serviced are in the City Codes form and the airplane type and seats available are in the Airplane Seat Information form.

The List Of Available Flights view displays the various flights and is categorized by the Available Flight Number.

Table 2 shows the fields in the List Of Available Flights form.

**Table 2 List of Available Flights Form**

| Name | Field Name | Type | Description |
|---|---|---|---|
| Flight Number | AvailableFlightNumber | Text | The number of the flight, which may be used on multiple days of the week but not on the same day |
| Departure Time | DepartureTime | Date/time | The time the flight leaves the departure city |
| Arrival Time | ArrivalTime | Date/time | The time the flight arrives in the arrival city |
| Duration | Duration | Number | The amount of hours between departure and arrival |
| Airline Name | AirlineName | Text | The name of the airline providing the flight |
| Departure City Code | DisplayOnlyDeparture | Dialog list based on city codes | The three letter code of the departure city |
| Arrival City Code | DisplayOnlyArrival | Dialog list based on city codes | The three letter code of the arrival city |
| AirplaneType | ListAirplaneType | Dialog list | The type of plane |
| Food | Food | Radio button -> (breakfast, lunch, dinner, snack, none) | Specifies whether breakfast, lunch, dinner, snack, or no food (none) will be provided on the flight |
| Scheduled Days | ScheduledDays | Checkbox -> Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday | The days the flight will be scheduled to fly |
| First Class Price | FirstClassPrice | Number | The price of a first class ticket |
| Coach Price | CoachPrice | Number | The price of a coach ticket |
| Departure City Code | DepartureCityCode | Text | Hidden |
| Arrival City Code | ArrivalCityCode | Text | Hidden |
| Display Status | DisplayStatus | Text | Hidden |

The Scheduled Flights form contains the flight information for a particular flight on a particular date. It is used to book and track seats on a specific flight on a specific date. All of the scheduled flight specific data is contained in this form.

The Scheduled Flights view displays the various scheduled flights and is categorized by the Flight By Date. The Processed Flights view displays all flights that have arrived or have no available seats to book.

Table 3 shows the fields in the Scheduled Flights form.

**Table 3 Scheduled Flights Form**

| Name | Field Name | Type | Description |
|---|---|---|---|
| Flight By Date | ScheduledFlightByDate | Text | The ID to be used to track a specific flight on a specific date, created by joining the flight number with the date of the flight |
| Flight Number | ScheduledFlightNumber | Text | The flight number |
| AirplaneType | ScheduledAirplaneType | Dialog list | The type of plane |

| | | | |
|---|---|---|---|
| First Class Seats Available | FirstClassSeatsAvailable | Number | The number of first class seats that are still available on the plane |
| Coach Seats Available | CoachSeatsAvailable | Number | The number of coach seats that are still available on the plane |
| Status | Status | Radio button -> (Cancelled, arrived, departed, standby, new) | The status of the flight |
| DepartureDate | ScheduledDepartureDate | Date | The date the flight is departing |
| ArrivalDate | ScheduledArrivalDate | Date | The date the flight is arriving, which may differ if it is an overnight flight or crossing the international date line |

The Airplane Seat Information form contains the airplane specific information. It is used to list the type of plane and the total number of seats available.

The Airplane Seat Information view displays the various airplanes and is categorized by the Airplane Type.

Table 4 shows the fields in the Airplane Seat Information form.

**Table 4 Airplane Seat Information Form**

| Name | Field Name | Type | Description |
|---|---|---|---|
| AirplaneType | SeatAirplaneType | Text | The type of plane |
| First Class Seats | FirstClassSeats | Number | The number of first class seats that exist on the plane |
| Coach Seats | CoachSeats | Number | The number of coach seats that exist on the plane |
| All Airplane Info | AllAirplaneInfo | Text | Hidden |

The Ticket Information form contains the information for one ticket. There will be one or more tickets included in an invoice.

The Active Tickets view displays the tickets that have not yet been processed. Once these orders are processed, they are removed from this view and displayed in either the Processed Tickets or the Inactive Tickets view, depending on whether the ticket was successfully processed.

The Inactive Tickets view displays the tickets that could not be completed. An example of tickets that could not be processed are for orders that were placed when no more tickets were available for the flight or for a flight that was cancelled.

The Processed Tickets view displays all the completed tickets.

Table 5 shows the fields in the Ticket Information form.

**Table 5 Ticket Information Form**

| Name | Field Name | Type | Description |
|---|---|---|---|
| Invoice Number | InvoiceNumber | Text | Number used to tie tickets together |
| Flight Number | TicketFlightNumber | Text | Ties to the list of available flights document |
| Ticket Class | TicketClass | Dialog List | Which class the ticket is: First Class or Coach |
| Seat Number | SeatNumber | Text | The seat number on this flight |
| Ticket Price | TicketPrice | Number | The price of the ticket for that seat |
| Paid Status | PaidStatus | Radio button | The status of the ticket |
| Customer Number | TicketCustomerNumber | Text | The number of the customer paying for the ticket |

| | | | |
|---|---|---|---|
| Ticket Status | TicketStatus | Dialog List | The status of the ticket: Active, Inactive, or Processed |
| Passenger First Name | PassengerFirstName | Text | The name of the customer sitting in the assigned seat. This may differ from the person purchasing the ticket and therefore the customer number |
| Passenger Middle Name | PassengerMiddleName | Text | The middle name of the passenger |
| Passenger Last Name | PassengerLastName | Text | The last name of the passenger |
| Passenger Street Address | PassengerStreet | Text | The address for the passenger |
| Passenger City | PassengerCity | Text | The passenger's city |
| Passenger State | PassengerState | Text | The passenger's state |
| Passenger Zip | PassengerZip | Text | The zip code for the passenger's address |
| Passenger Country | PassengerCountry | Text | The passenger's country |
| Passenger Phone | PassengerPhone | Text | The phone number for the passenger |
| Flight By Date | FlightByDate | Text | The flight by date number for the flight of this ticket |

The Credit Information form contains the credit information for the customer as identified by the customer number.

The Credit Information view displays the credit card information used by the customer to pay for their flight and is categorized by the Customer Number.

Table 6 shows the fields in the Credit Information form.

**Table 6 Credit Information Form**

| Name | Field Name | Type | Description |
|---|---|---|---|
| Card Type | CardType | Dialog list (Mastercard, Visa, Diner's Club) | The type of credit card |
| Credit Card Number | CreditCardNumber | Text | The number of the card |
| Expiration Date | ExpirationDate | Date / time | The date the card expires |
| Customer Number | CreditCustomerNumber | Text | The customer number associated with that card |
| Invoice Number | InvoiceNumber | Text | Number used to tie tickets together |

The City Codes form contains the list of cities supported and their corresponding three letter city codes.

The City Codes view displays the various city codes and is categorized by the code.

Table 7 shows the fields in the City Codes form.

**Table 7 City Codes Form**

| Name | Field Name | Type | Description |
|---|---|---|---|
| City | City | Text | The city |
| State | State | Text | The state or providence |
| Country | Country | Text | The country |
| Code | Code | Text | The three letter code to represent the city specified |
| All City Code Info | AllCityCodeInfo | Text | Hidden |
| Save Options | SaveOptions | Number | Hidden - set to 0 so default doc will not save |

The Bill Information form contains information for billing customers.

The Bill Information view displays the ticket information used to bill a customer for their flight and is categorized by the Customer Number.

Table 8 shows the fields in the Credit Information form.

**Table 8 Credit Information Form**

| Name | Field Name | Type | Description |
|---|---|---|---|
| Bill Price | BillAmount | Number | The amount the customer owes |
| Customer Number | BillCustomer | Text | The customer number |
| Ticket Invoice Info | BillTicketInfo | Text | The ticket to which the bill is associated |

The viewCustomerFldNumber, viewFlightFldNumber, viewInvoiceFldNumber, viewSeatFldNumber, and viewStatementFldNumber are hidden views that display the customer number, flight number, invoice number, seat number, and statement number documents. When a new document is created, the PostOpen event uses the current value in the document to create a unique number. Once the number has been created, the PostOpen event increments the value and stores it in the document.

In addition to the forms and views created within their application database, the flights company also added the following form and view to the Names and Address book on their Lotus Domino server, shown in Table 9.

**Table 9 Lotus Domino Forms and Views**

| Lotus Domino forms | Lotus Domino views |
|---|---|
| Flight Person | Flights Customers |

The Flight Person form contains the name, address information, and customer number for each of the flights customers. The flights company based this form off of the existing Person form from the Address Book design and modified it to better fit their needs.

The Flight Customers view displays the customer information and is categorized by the Customer Number.

Table 10 shows the fields in the Flight Person form.

**Table 10 Flight Person Form**

| Name | Field Name | Type | Description |
|---|---|---|---|
| First name | FirstName | Text | The customer's first name |
| Middle initial | MiddleInitial | Text | The customer's middle initial |
| Last name | LastName | Text | The customer's last name |
| Internet Address | InternetAddress | Text | The customer's internet address |
| Internet Password | HTTPPassword | Text | The customer's password for accessing the flights company's website |
| Street Address | StreetAddress | Text | The customer's home street address |
| City | City | Text | The customer's city |
| State/Province | State | Text | The customer's state/province |
| Zip/Postal Code | Zip | Text | The customer's zip/postal code |
| Country | Country | Text | The customer's country |
| Home Phone | PhoneNumber | Text | The customer's home phone number |
| Customer Number | PersonalID | Number | The customer number |

# Lotus Domino agents

Lotus Domino agents are design elements added to a Lotus Domino database to automate tasks. Agents can be initiated by a user action or run on a scheduled basis. Agents are commonly used to update or create documents, or to access data from the Lotus Domino database or other sources. Lotus Domino agents can be written in Java, LotusScript, or Formula Language.

The creation of agents requires Lotus Domino Designer. When you create an agent, you specify when you want it to run, what language the code will be written in, and what documents it runs under. After you have written the code and compiled it, it is automatically scheduled to run at the time you previously specified. While you are writing code, you can make use of the built-in debugging capabilities of Lotus Domino Designer to help you debug your code.

The following provides detailed information about the Lotus Domino agents written for the Lotus Domino flight application:

- Check Unique City Code
  This LotusScript agent receives a city code as input from a Web browser using the City Codes form. This agent uses that information along with information from the existing City Code documents (created from the City Codes form) to search through all the City Code documents. If a match is found, an error message is displayed and the user is notified and allowed to create a different City Code document. If no match is found, the City Code document will be created.
- Create Random Customer Number, Flight Number, and Invoice Number - Web Only
  These LotusScript agents generate a unique customer number, flight number, or invoice number for new customers, flights, or invoices added through the Lotus Domino Web interface.
- Delete Button City Codes and List of Available Flights
  This Java agent runs based on the user clicking on the Delete action button in the City Code or Flight views. It marks the selected City Code or Flight document as deleted.
- Delete Selected City Codes
  This Java agent runs on a scheduled basis. It will delete any city codes that have a CityCodeStatus of Delete from the City Codes documents. It will also delete the documents from List Of Available Flights, Scheduled Flights and Tickets that match the city code.
- Delete Selected Flights
  This Java agent runs on a scheduled basis. It will delete any flights that have a DisplayStatus of Delete from the List Of Available Flights documents. It will also delete the documents from Scheduled Flights and Tickets that match the flight number.
- Delete Ticket - Update Available Seats
  This Java agent runs based on the user clicking an action button. It will increment the number of available seats based on the FlightByDate and Ticket Class selected from the Scheduled Flight document.
- Generate Bill
  This Java agent runs based on the user clicking an action button. It will generate the billing information for the customer based on the customer number selected.
- Generate Price and Seat Number
  This Java agent runs based on the user clicking an action button. It will generate the price information and calculate the next available seat for a particular flight and seat class based on the FlightByDate and Ticket Class selected.
- Populate Scheduled Flights
  This Java agent runs on a scheduled basis. It creates Scheduled Flights documents with information gathered from the List of Available Flights and Airplane Seat Information documents where the display status is "New". Each flight will be populated for one month from today's date.
- Scheduled Flight Status
  This Java agent runs on a scheduled basis. It will change the status of flights from new to departed and from departed to arrived based on the Date and Time of the flight gathered from the List of Available Flights and Scheduled Flights documents.
- Update Scheduled Flights
  This Java agent runs on a scheduled basis. It checks the status of flights and if it finds any flights that have

departed from the previous run of the agent, it updates the tickets associated with the flight by moving them to the Processed Tickets view.

- Update Ticket Status
  This Java agent runs on a scheduled basis. It will change the status of tickets from active to processed or inactive based on the Date and Time of the flight gathered from the the List of Available Flights and Scheduled Flights documents. A ticket becomes inactive if a flight is cancelled. Processed tickets are those that have departed.

The source code for the Generate Price and Seat Number and the Check Unique City Code agents are shown in the ***Example: Source code for Lotus Domino agents*** section.

## Application details

From the Lotus Domino Web interface, the flights employee can perform several tasks. These tasks fall under four main categories of customer, flights, flight reservations, and billing. Each of these categories contain actions that the employee can perform. These actions are as follows:

- Customer
  A flights employee can add a customer by entering the customers name, password, internet address, and home address. The information entered along with the generated customer number is stored in the Lotus Domino LDAP directory. Once created, this new document can then be updated, deleted, or displayed.
- Flights
  - A flights employes can add a flight by entering departure and arrival time, length of flight, departure and arrival city code, airplane type, type of food available, scheduled flight days, and first class and coach price. The information entered is stored in the List Of Available Flights form. Once created, this new document can then be displayed or deleted.
  - A flights employee can also add a city code by entering city, state, country, and city code. The information entered is stored in the City Codes form after the city code is checked to make sure it is unique. If it is unique, the document is created. Once created, the document can be displayed or deleted.
- Flight Reservations
  A flight employee can book a flight by entering scheduled flight by date, class of ticket (coach/first class), generated price, generated seat number, paid status, customer number, and passenger information. The information entered is stored in the Ticket form. Once created, the document can be displayed in the following states: active, processed, or inactive.
- Billing
  A flight employee can generate a bill for a ticket by entering credit card type, number, expiration date, and the customer number. The information entered is stored in the Credit Information form. Once created, the document can be displayed or updated.

# Lotus(R) Domino(TM) environment key findings

Following is a list of key findings that we uncovered while creating and using the flights scenario Lotus Domino environment.

- In several flight documents, we used an agent to generate a unique number. This agent was called by the WebQueryOpen event. The agent would generate the number when a new document was created, but the generated number would not save when the document was saved. The following excerpt from Lotus Domino Designer help text explains how the WebQueryOpen event works and why this value was not being saved:

  "WebQueryOpen agents run when the user opens a form or document, but do not run when the user saves a document. This means that computed fields set by a WebQueryOpen agent are not saved when the user submits a document. To make sure computed fields are saved, you can either recalculate them in the WebQuerySave agent or set the form property 'Generate HTML for all fields'."

  After selecting the form property 'Generate HTML for all fields', the computed number was saved along with all of the other fields in the document.

- To display a view after submitting a document through the Web, instead of displaying the default text: "Form processed", do the following:
  - Create a hidden text field in the form that is computed for display
  - Name the field $$Return
  - Use a formula similar to this in the field value:
    "[/"+@Subset(@DbName;-1)+"/YourViewName?OpenView&DblClkTarget=_self]"

    If your view name has spaces in it, use a '+' instead of a space. (i.e. Your+View+Name)
    If your view name is categorized under other views, use a double '\\'. (i.e. YourView\\ViewName)

- We experienced poor performance and other odd behavior when we had the view property 'Use applet in the browser' selected under the For Web Access in the advanced tab of the view properties.

- In the Flight Person form, the form property 'Generate HTML for all fields' needed to be selected to allow the data to be maintained in the form when entering data on the Web. Without it selected, when clicking on the next tab in the customer form, the data in the other tabs would not be maintained. With it selected, the data was maintained as each of the tabs was selected.

- We wanted to only allow unique city codes to be saved when creating a new unique city code from the Web.

  The following excerpt from the Lotus Domino Designer help text explains how the WebQuerySave event works when checking for unique values:

  "Simulating CGI programs that run on user-supplied data by programming a WebQuerySave event and adding a SaveOptions field with a value of '0' to the form. When the agent runs, you can collect field values from the filled-out form without generating a new Notes(TM) document."

  Having the SaveOptions set to 0 will keep the document from being saved. As soon as this value is set to 1, the document is saved. There is no need to perform an if check on SaveOptions in the submit button. Lotus Domino handles the saving based on the value of the SaveOptions field.

  For example, here is the solution we used:

  1. In the Form, create a Submit button and also a hidden field called SaveOptions with an initial value of 0
  2. In the Submit button, code the following formula:
     @Command([FileSave]);
     @Command([FileCloseWindow])

3. In the WebQuerySave event, enter the formula that runs an agent
4. In the agent, add the following code:

```
If (foundCityCode) Then
Print "<SCRIPT LANGUAGE=JavaScript(TM)>"
Print "alert(""Duplicate City Code found. Please enter a new city code."")"
Print "location.href = ""../../" + file + "/City+Codes?OpenForm"""
Print "</SCRIPT>"
Else
'// The following line will set the SaveOptions on the document as 1 which will cause Notes to save the document.
Set item = doc.ReplaceItemValue("SaveOptions", 1)
End If
```

# References


- Lotus Domino Release 5.0: A Developer's Handbook, IBM Redbook(R) SG24-5331-01
- IBM Lotus Domino for iSeries(TM) - OS/400(R) Web site
  http://www.ibm.com/servers/eserver/iseries/domino/
- IBM Lotus Domino for iSeries (PartnerWorld(R) for Developers)
  http://www.as400.ibm.com/developer/domino/
- Lotus Web site
  http://www.lotus.com

# Example: Source code for Lotus Domino agents

This section contains source code for two of the flight's application Lotus (R) Domino (TM) agents. The first example is Generate Price and Seat Number agent which is written in Java(TM). This agent generates price information and calculates the next available seat. The second example is Check Unique City Code agent which is written in LotusScript. This agent verifies that the city code is unique.

**Example 1:** Generate Price and Seat Number agent

```
////////////////////////////////////////////////////////////////////////
/*
* This Java Agent will run based on the user clicking an action button.
* It will generate the price information and calculate the next available seat
* based on the FlightByDate and Ticket Class selected.
*
* Scenario Name: TFC Flights
*
* Java Version: JDK 1.1.8
*/
////////////////////////////////////////////////////////////////////////

import lotus.domino.*;

public class JavaAgent extends AgentBase {

public void NotesMain() {

System.out.println("Starting: Generate Price and Seat Number agent.");

try
{
Session session = getSession();
AgentContext agentContext = session.getAgentContext();

generateSeat(session, agentContext);

System.out.println("Done: Generate Price and Seat Number agent.");
}
catch(Exception e) {
e.printStackTrace();
}

} // end Notes(TM) main


public void generateSeat(Session session, AgentContext agentContext)
{
// get the Flight by date and ticket class data from the current document
boolean isFirstClass = false; //initially set this flag to false (i.e. coach)

try
{
Database db = agentContext.getCurrentDatabase();
DocumentCollection collection = agentContext.getUnprocessedDocuments();
if (collection.getCount() < 1)
{
// there was a problem accessing the current doc, quit agent
System.out.println("Error with the current document, agent ending.");
return;
} // end if
else
{
Document doc = collection.getFirstDocument();
String flightByDate = doc.getItemValueString("FlightByDate");
String ticketClass = doc.getItemValueString("TicketClass");
DocumentCollection scheduledFlightDC = db.search("SELECT ((Form = \"Scheduled Flights\")) &
((@Contains(ScheduledFlightByDate; \"" + flightByDate + "\")))");
if (scheduledFlightDC.getCount() < 1)
{
```

```
// invalid flight by date specified - no matching flight by date found
System.out.println("Invalid flight by date specified - no matching flight by date found,
agent ending.");
return;
} // end if
else
{
int seatsRemaining = 0;
int seatNumber = 0;
Document scheduledFlightDoc = scheduledFlightDC.getFirstDocument();

if (ticketClass.equals("First Class"))
{
isFirstClass = true;

seatsRemaining = scheduledFlightDoc.getItemValueInteger("FirstClassSeatsAvailable");

//check to see if a seat is still available
if (seatsRemaining >= 1)
{
// get the next available seat number
seatNumber = getNextAvailableSeat(db, isFirstClass,
scheduledFlightDoc.getItemValueString("ScheduledAirplaneType"));
int nextAvailableSeat = seatNumber - seatsRemaining + 1;

String strObj = String.valueOf(nextAvailableSeat);
doc.replaceItemValue("SeatNumber", strObj);

int price = getPrice(flightByDate, session, agentContext, isFirstClass); //get the price info
for this ticket
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);

// a seat is available
Integer intObject = new Integer(seatsRemaining-1); //decrement num of available seats
scheduledFlightDoc.replaceItemValue("FirstClassSeatsAvailable", intObject);
scheduledFlightDoc.save(true, true);
} //end if seats remaining >= 1

else
{ //if seats remaining = 0, check to see if any tickets have been deleted
String deletedSeats = scheduledFlightDoc.getItemValueString("DeletedFirstClassSeats");
if (deletedSeats != null)
{ // there's a cancelled seat(s) available, book the seat
int index = deletedSeats.indexOf(";");
String availSeat = deletedSeats.substring(0, index);
Integer intObject = Integer.valueOf(availSeat);
// update the deleted seats, removing the seat that was just booked and leaving the remaining
as is
scheduledFlightDoc.replaceItemValue("DeletedFirstClassSeats", deletedSeats.substring(index +
1));
scheduledFlightDoc.save(true, true);

String strObj = String.valueOf(availSeat);
doc.replaceItemValue("SeatNumber", strObj);

int price = getPrice(flightByDate, session, agentContext, isFirstClass); //get the price info
for this ticket
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);

} // end if deletedSeats != null
else
{
// the flight is full
System.out.println ("<SCRIPT LANGUAGE=JavaScript(TM)>");
System.out.println ("alert(\"No more seats are available on this flight. Please select a new
flight.\")");
```

```
System.out.println ("</SCRIPT>");
} // end else

} // end seats remaining = 0

} // end if first class
else
{ // the ticket is for a coach seat
seatsRemaining = scheduledFlightDoc.getItemValueInteger("CoachSeatsAvailable");

//check to see if a seat is still available
if (seatsRemaining >= 1)
{
// get the next available seat number
seatNumber = getNextAvailableSeat(db, isFirstClass,
scheduledFlightDoc.getItemValueString("ScheduledAirplaneType"));
int nextAvailableSeat = seatNumber - seatsRemaining + 1;

String strObj = String.valueOf(nextAvailableSeat);
doc.replaceItemValue("SeatNumber", strObj);

int price = getPrice(flightByDate, session, agentContext, isFirstClass); //get the price info
for this ticket
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);

// a seat is available
Integer intObject = new Integer(seatsRemaining-1); //decrement num of available seats
scheduledFlightDoc.replaceItemValue("CoachSeatsAvailable", intObject);
scheduledFlightDoc.save(true, true);
} //end if seats remaining >= 1

else
{ //if seats remaining = 0, check to see if any tickets have been deleted
String deletedSeats = scheduledFlightDoc.getItemValueString("DeletedCoachSeats");
if (deletedSeats != null)
{ // there's a cancelled seat(s) available, book the seat
int index = deletedSeats.indexOf(";");
String availSeat = deletedSeats.substring(0, index);
Integer intObject = Integer.valueOf(availSeat);
// update the deleted seats, removing the seat that was just booked and leaving the remaining
as is
scheduledFlightDoc.replaceItemValue("DeletedCoachSeats", deletedSeats.substring(index + 1));
scheduledFlightDoc.save(true, true);

String strObj = String.valueOf(availSeat);
doc.replaceItemValue("SeatNumber", strObj);

int price = getPrice(flightByDate, session, agentContext, isFirstClass); //get the price info
for this ticket
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);

} // end if deletedSeats != null
else
{
// the flight is full
System.out.println ("<SCRIPT LANGUAGE=JavaScript>");
System.out.println ("alert(\"No more seats are available on this flight. Please select a new
flight.\")");
System.out.println ("</SCRIPT>");
}
} // end seats remaining = 0
} // end else ticket is coach
}
} // end else
} // end try
```

```
catch (NotesException ne) {
ne.printStackTrace();
}
catch (Exception e) {
e.printStackTrace();
}

} // end method generateSeat

public int getNextAvailableSeat(Database db, boolean isFirstClass, String airplaneType)
{
// go to Airplane Seat Info view and find the matching plane type
int seatNum = 0;
try
{
DocumentCollection seatDC = db.search("SELECT ((Form = \"Airplane Seat Information\")) &
((@Contains(SeatAirplaneType; \"" + airplaneType + "\")))");
if (seatDC.getCount() < 1)
{
// no matching airplane type found
System.out.println("Invalid airplane model specified - no matching record found, agent
ending.");
return 0;
} // end if
else
{
Document seatDoc = seatDC.getFirstDocument();
if (isFirstClass)
{
seatNum = seatDoc.getItemValueInteger("FirstClassSeats");
}
else
{
seatNum = seatDoc.getItemValueInteger("CoachSeats");
}

} // end else
} // end try
catch (NotesException ne) {
ne.printStackTrace();
}
catch (Exception e) {
e.printStackTrace();
}

return seatNum;
} // end method getNextAvailableSeat

public int getPrice(String flightByDate, Session session, AgentContext agentContext, boolean
isFirstClass)
{
int price = 0;
try
{
// calculate flight number
int index = flightByDate.lastIndexOf("_");
String flightNumber = flightByDate.substring(0, index);

Database db = agentContext.getCurrentDatabase();
DocumentCollection flightDC = db.search("SELECT ((Form = \"List Of Available Flights\")) &
((@Contains(AvailableFlightNumber; \"" + flightNumber + "\")))");

if (flightDC.getCount() < 1)
{
// there was a problem accessing the current doc, quit agent
System.out.println("Error with the current document, agent ending.");
return 0;
} // end if
else
{
```

```
Document doc = flightDC.getFirstDocument();
if (isFirstClass)
{
price = doc.getItemValueInteger("FirstClassPrice");
}
else
{
price = doc.getItemValueInteger("CoachPrice");
}

} // end else

} // end try
catch (NotesException ne) {
ne.printStackTrace();
}
catch (Exception e) {
e.printStackTrace();
}

return price;
} // end method generateSeat
} // end class
```

**Example 2:** Check Unique City Code Agent

```
Sub Initialize

'//=============================================================================================
'// This Domino agent receives a City Code as input from a browser using the City Codes Form.
'// This agent uses that information along with information from the existing City Code documents (created from the City Codes form).
'// This agent uses the veiw 'City Codes' to search through all the City Code documents. If a match is found, the user is
'// notified and allowed to create a new City Code document. If no match is found, the City Code document will be created.
'//=============================================================================================

'// Set foundCityCode variant
Dim foundCityCode As Variant
foundCityCode = False

'// Create a notes session
Dim session As New NotesSession

Messagebox "Running agent " & session.CurrentAgent.Name & " in database " & session.CurrentDatabase.Title & " as " &
session.CommonUserName

'// Open this database
Dim db As NotesDatabase
Set db = session.CurrentDatabase
If Not ( db.IsOpen) Then
Messagebox "Database " & session.CurrentDatabase.Title & " did not open. Agent ending."
Exit Sub
End If

'// Set doc equal to the current session transient values (i.e. the city code document code variables)
Dim doc As NotesDocument
Set doc = session.DocumentContext
Dim unid As String
Dim file As String
file = db.FileName

'// Set the view to the City Codes view, to locate a City Codes document
Dim CCview As NotesView
Dim CCdoc As NotesDocument
Set CCview = db.GetView("City Codes")
Set CCdoc = CCview.GetFirstDocument

'// Search through the City Code documents until a match is found or no match is found
While ((Not (CCdoc Is Nothing)) And (Not foundCityCode))
If (CCdoc.Code(0) = doc.Code(0)) Then
unid = CCdoc.UniversalID
```

```
foundCityCode = True
Else
Set CCdoc = CCview.GetNextDocument (CCdoc)
End If
Wend

'// If match found, display a message and allow user to create a new City Code document
'// If no match found, allow user to create a new City Code document
If (foundCityCode) Then
Print "<SCRIPT LANGUAGE=JavaScript>"
Print "alert(""Duplicate City Code found. Please enter a new city code."")"
Print "location.href = ""../../" + file + "/City+Codes?OpenForm"""
Print "</SCRIPT>"
Else
On Error Goto Errhandle
'// Call doc.Save(True, True)
'// Print "<SCRIPT LANGUAGE=JavaScript>"
'// Print "location.href = ""../../" + file + "/City+Codes/doc?SaveDocument"""
'// Print "alert(""Saving."")"
'// Print "location.href = ""../../" + file + "/City+Codes?OpenView"""
'// Print "</SCRIPT>"
Set item = doc.ReplaceItemValue("SaveOptions", 1)
End If

Errhandle:
' Use the Err function to return the error number and
' the Error$ function to return the error message.
Messagebox "Error" & Str(Err) & ": " & Error$
Exit Sub

End Sub
```

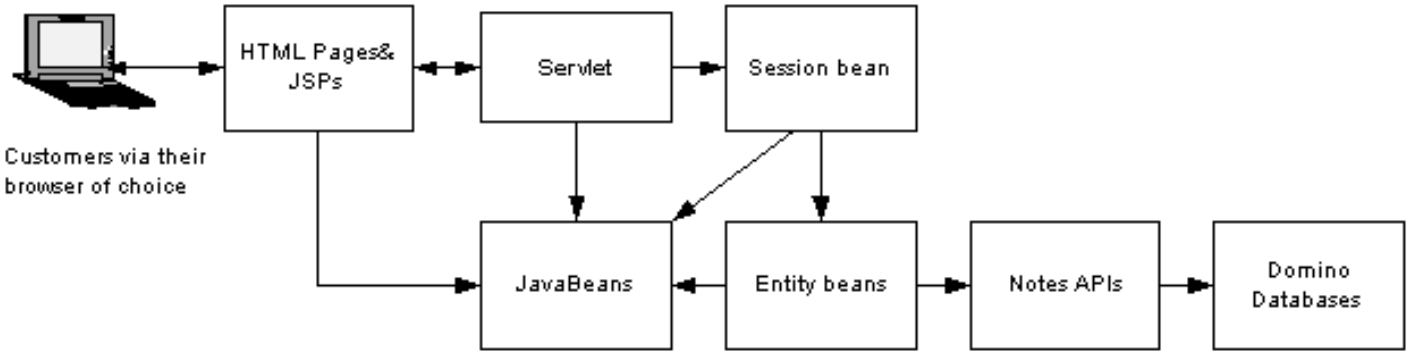
# WebSphere(R) Application Server environment overview

The WebSphere Application Server environment was set up during phase two of the iSeries(TM) System Test flights scenario work. During phase two, the flight company's goal was to allow their customers to book flights from their Web site. They decided to build this interface using HTML pages, Java(TM) Server Pages (JSPs), JavaBeans(TM), enterprise beans, and servlets. This interface allowed customers to view and book flights on-line, retrieve flight information, update customer information, and pay bills on-line.

## Application Model

The flights application uses the application model depicted in Figure 1. In this model, a browser accesses the Java Server Pages (JSPs) indirectly through a servlet which interacts with the business logic using enterprise beans. The enterprise beans extract the needed information from the Lotus(R) Domino(TM) database. After receiving the client request, the servlet performs any necessary computation and creates the JavaBeans. The JSP is invoked with the appropriate JavaBeans. The JSP extracts the information it requires from the JavaBeans and merges them with the HTML page. The browser then interprets and renders the HTML.

**Figure 1 Flights Application Model**




# Application Process Flow

There are a number of HTML and JSP pages used within the flights application. Table 1 contains the list of HTML pages and Table 2 contains the list of JSPs. The application flow is displayed in Figure 2.

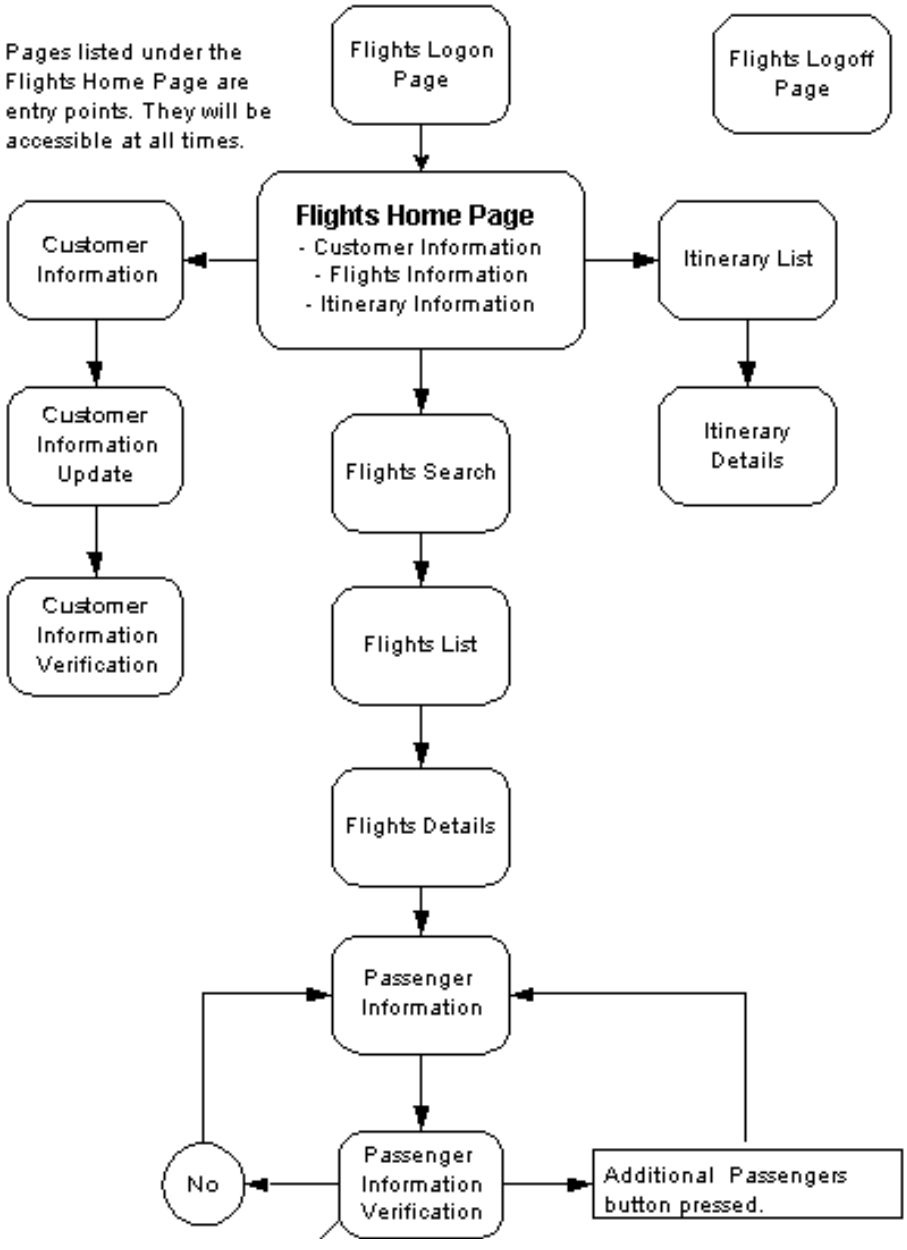
**Table 1 Flights Application HTML pages**

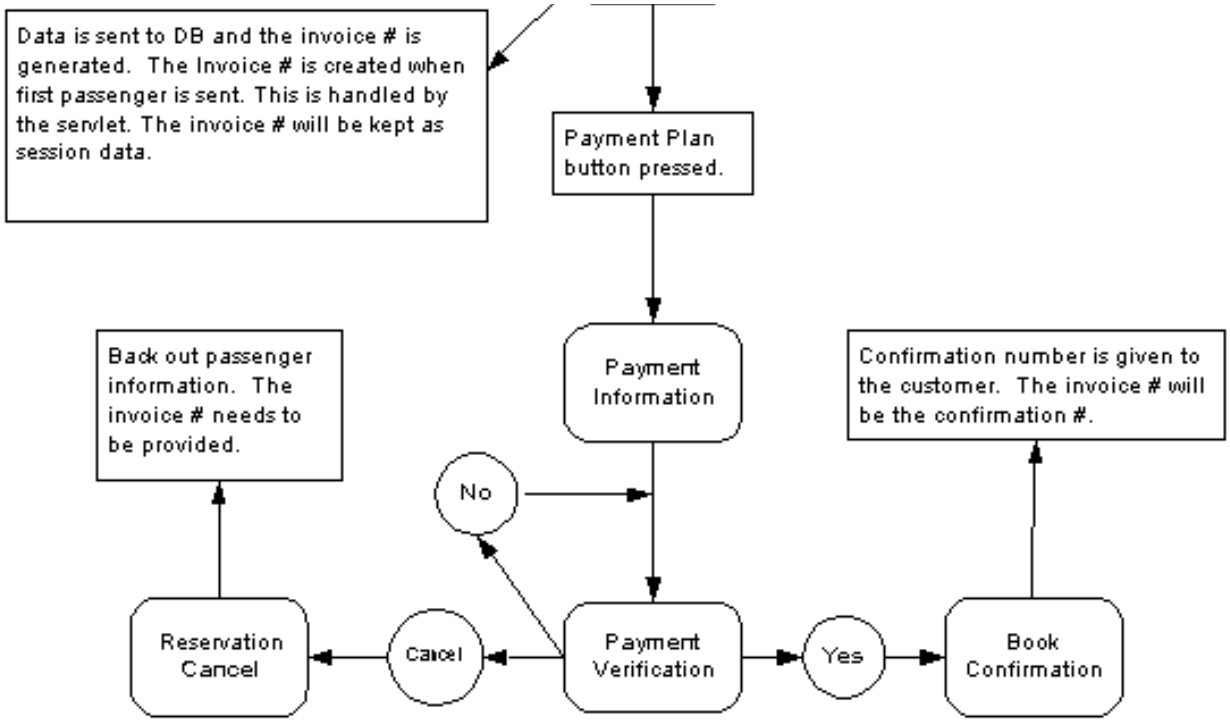
| HTML page | Functions |
|---|---|
| homePage | Allows the customer to choose between displaying the following information:<br>• flight<br>• customer<br>• itinerary |
| index | Brings together the menu, top, and homePage frames |
| logOn | Allows customer to log on |
| menu | Provides menu options on the left side |
| top | Provides top of page which contains the logo |

**Table 2 Flights Application JSPs**

| JSP | Functions |
|---|---|
| customerInformation | Displays information about the customer |
| logOff | Allows customer to log off |
| customerInformationUpdate | Allows customer to update their information |
| customerInformationConfirmation | Allows customer to verify their updated information |
| flightsSearch | Provides list of criteria for searching for flights |
| flightsList | Provides a list of flights that match the search criteria |
| flightsDetails | Provides the detailed flight information for a specific flight |
| passengerInformation | Allows customer to enter passenger information |
| passengerInformationVerification | Allows customer to verify passenger information |
| paymentInformation | Allows customer to enter payment information |
| paymentInformationVerification | Allows customer to verify payment information |
| reservationCancel | Allows customer to cancel the reservation |
| bookConfirmation | Confirms the booking |
| itineraryList | Provides the list of itineraries for the customer |
| itineraryDetails | Provides the detailed itinerary information for a specific itinerary |

**Figure 2 Flights Application Flow**






## Development Environment

The following products were used during the development of this application:

- WebSphere Studio
- Visual Age for Java Enterprise Edition
- WebSphere Studio Application Developer

The JSPs were initially developed using WebSphere Studio and the enterprise beans were developed using Visual Age for Java. In the middle of development, we migrated to WebSphere Application Server Version 4.0. This change prompted us to move our development of the JSPs and enterprise beans to WebSphere Studio Application Developer.

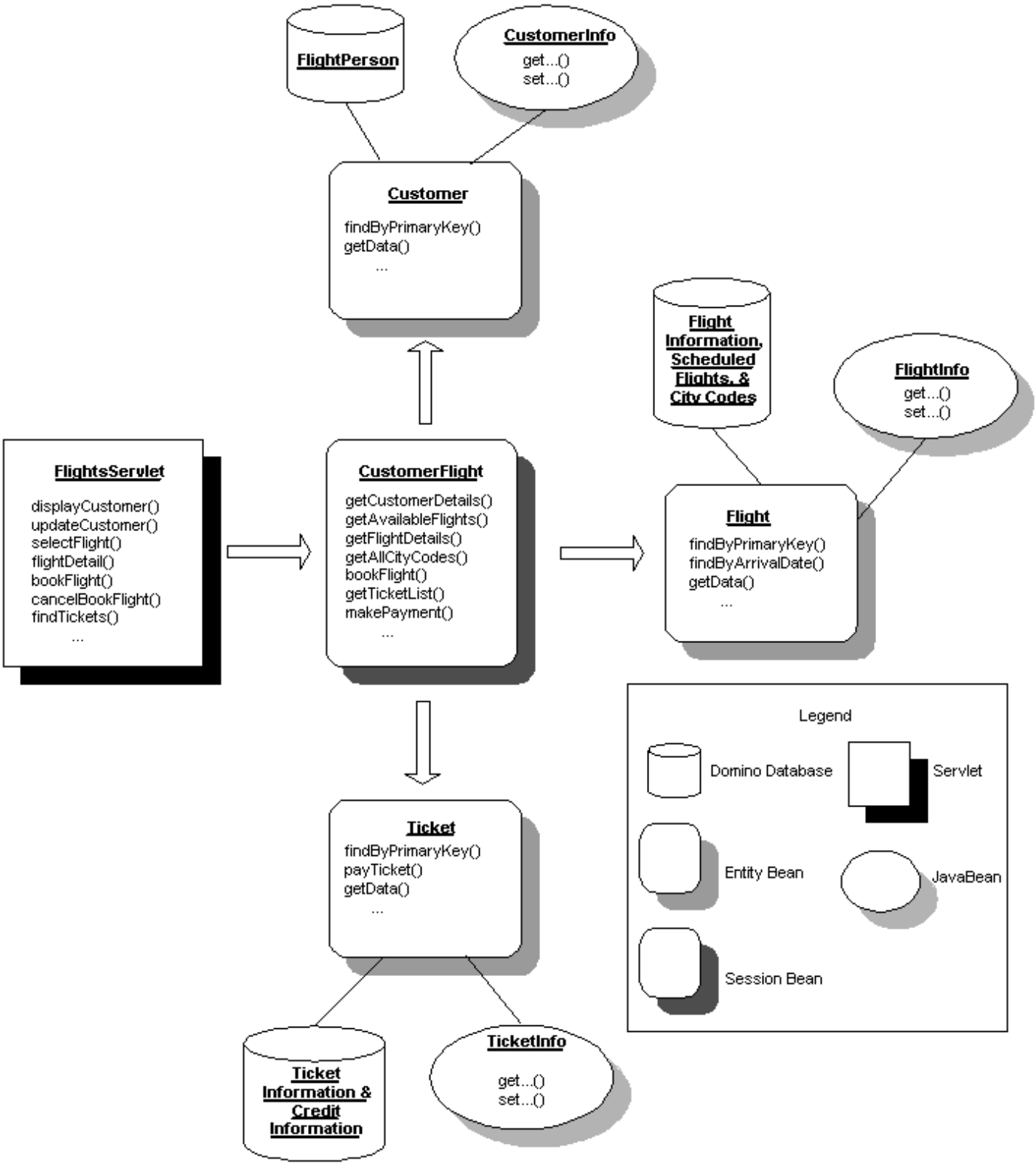
# WebSphere(R) Application Server environment application flow

## Application Details

This section provides details on the flights application model including the use of servlets, JavaBeans(TM), enterprise beans, and JSPs. There is one servlet (FlightsServlet) that acts as the main controller. All requests are sent to this servlet, which then performs the requested task. The servlet accesses a session bean (CustomerFlight) to accomplish the majority of its work. The CustomerFlight session bean uses entity beans (Customer, Flight, and Ticket) to accomplish the desired task.

Figure 1 illustrates the relationship between the servlet and enterprise beans.

**Figure 1 Application relationship**



# Design Considerations

The initial plan was to use a Lotus(R) Domino(TM) JDBC Driver within our enterprise beans to access the Lotus Domino databases. However, in WebSphere Application Server, you cannot create a datasource that uses the Lotus Domino JDBC driver on the iSeries(TM). Since the iSeries has not yet ported the Lotus Domino JDBC driver, we also could not implement our own connection pooling. A Lotus Domino JDBC driver for iSeries is scheduled to be available in a future release. Because of the limitations listed above, we used the Lotus Domino APIs to access the Lotus Domino database within our entity beans.

When the application was designed, the decision was made to use bean-managed persistence entity beans. We needed to use bean-managed entity beans because the entity beans would be using Lotus Domino APIs to access Lotus Domino databases.

# Flights Servlet

The flights application uses the FlightsServlet to control the flow of the application. The FlightsServlet is used to provide the Flights customers with the capability to view and update customer data, view scheduled flight data, and book flights using a Web browser. The JSP pages called by this servlet are used for the presentation of data. They perform a minimal amount of processing work. The majority of the processing work is done by the FlightsServlet through the following methods:

- doPost
  - Creates a session between the Web server and the browser making the request.
  - Examines each request and routes it to the appropriate method within the servlet based on the jspRequest value.
- doGet
  - Creates a session between the Web server and the browser making the request.
  - Sets the jspRequest value accordingly and calls doPost().
- errorHandle
  - Handles any exceptions encountered by the servlet or enterprise beans.
  - Provides a message to the user on the error encountered.
- init
  - Called by the server immediately after the server constructs the servlet's instance.
  - Creates and looks up the CustomerFlight home object.
- bookFlight
  - Uses the CustomerFlight enterprise bean to book a flight for the specified customer.
  - Displays the passengerInformation.jsp.
- cancelBookFlight
  - Uses the CustomerFlight enterprise bean to cancel a flight that was being booked for a specific customer.
  - Displays the reservationCancel.jsp.
- confirmFlight
  - Uses the CustomerFlight enterprise bean to confirm and make payment for a booked flight.
  - Displays the bookConfirmation.jsp.
- displayCustomer
  - Uses the CustomerFlight enterprise bean to obtain the information for the specified customer.
  - Displays the customerInformation.jsp.
- displayJSP
  - Calls a JSP page. All the data that the JSP needs will be in the session.
- findTickets
  - Uses the CustomerFlight enterprise bean to obtain a list of tickets.
  - Displays the itineraryList.jsp.
- flightDetail
  - Uses the CustomerFlight enterprise bean to obtain the detailed flight information.
  - Displays the flightsDetails.jsp.
- flightList
  - Uses the CustomerFlight enterprise bean to obtain a list of all of the city codes.
  - Displays the flightSearch.jsp.
- getCustomerFlight
  - Creates a CustomerFlightHome object by looking up the CustomerFlightHome class.
- getInitialContext
  - Obtains an initial context for the specified URL.
- itineraryDetails
  - Uses the CustomerFlight enterprise bean to obtain the details of the itinerary and flight based on the itinerary number and flight by date.
  - Displays the itineraryDetails.jsp.
- selectFlight
  - Uses the CustomerFlight enterprise bean to obtain a list of all flights based on the search criteria provided.
  - Displays the flightsList.jsp.

- updateCustomer
    - Uses the CustomerFlight enterprise bean to update the customer data.
    - Displays the customerInfoConfirmation.jsp.

# Enterprise beans

Within the flights application, FlightsServlet acts as the main controller. All requests are sent to this servlet which then does the requested task. The servlet accesses a session bean (CustomerFlight) to accomplish the majority of its work. The CustomerFlight session bean uses entity beans (Customer, Flight, and Ticket) to accomplish the desired task.

## CustomerFlight enterprise bean

The CustomerFlight enterprise bean is a stateless session bean. It supplies methods that accomplish tasks that would be performed by a customer, for example:

- View or update customer information
- Obtain a list of available flights
- View itinerary details
- Book a flight
- Make a payment for a booked flight

The methods in the CustomerFlight session bean are described here:

- The bookFlight() method returns a TicketInfo object that contains the information for the ticket that was booked. The following tasks are performed:
    1. It checks to see if an invoice number is specified. If it is not specified, it will create a new ticket with a new invoice number. If the invoice number is specified, it will create a new ticket using the specified invoice number. The tickets are created using the Ticket enterprise bean.
    2. The getData() method is invoked in the Ticket enterprise bean and the data is stored in a TicketInfo object.
    3. The TicketInfo object is returned.
- The cancelBookFlight() method cancels the tickets for the specified invoice number. The following tasks are performed:
    1. It uses the findByInvoiceNumber() method within the Ticket enterprise bean to obtain a list of tickets with the specified invoice number.
    2. The list of tickets is then processed through and the remove() method is called on each Ticket to remove the documents from the Lotus Domino database.
- The getAllCityCodes() method obtains a list of all the city codes available in the City Codes document. The following tasks are performed:
    1. It selects all city codes from the City Codes form and the information is stored in a FlightInfo object.
    2. The FlightInfo object is returned.
- The getAvailableFlights() method returns a FlightInfo object that contains a list of all the flights that match the specified search criteria. The following tasks are performed:
    1. The parameter values are checked and a find by is performed based on the search criteria using the Flight enterprise bean.
    2. The getData() method in the Flight enterprise bean is invoked and the data is stored in a FlightInfo object.
    3. The FlightInfo object is returned.
- The getConnection() method returns a connection to the Lotus Domino database using Lotus Domino APIs. The following tasks are performed:
    1. The environment variables are used to create the connection.
    2. A new session is created to the Lotus Domino database.
    3. The database object is created and returned.
- The getCustomerDetails() method returns a CustomerInfo object that contains the customer information for the specified customer number. The following tasks are performed:
    1. A find by primary key is executed based on the customer number using the Customer enterprise bean.
    2. The getData() method is invoked in the Customer enterprise bean and the data is returned in a CustomerInfo object.
- The getFlightDetails() method returns a FlightInfo object that contains the flight information for the specified flight by date. The following tasks are performed:
    1. A find by primary key is executed based on the flight by date using the Flight enterprise bean.
    2. The getData() method is invoked in the Flight enterprise bean and the data is returned in a FlightInfo object.
- The getInitialContext() method returns the initial context for creating the entity beans.
- The getTicketList() method returns a TicketInfo object that contains a list of all the tickets based on a customer number or an invoice number. The following tasks are performed:
    1. A find by customer number or a find by invoice number is executed based on the parameter passed in using the Ticket enterprise bean.
    2. The getData() method is invoked in the Ticket enterprise bean and the data is stored in a vector.
    3. The vector is stored in a TicketInfo object.
    4. The TicketInfo object is returned.
- The makePayment() method returns a TicketInfo object. This method is used to make a payment for a booked flight. It will update all of the tickets with the same invoice number. The following tasks are performed:
    1. A find by invoice number is executed based on the invoice number using the Ticket enterprise bean.
    2. The payTicket() method is invoked in the Ticket enterprise bean.
    3. The getData() method is invoked in the Ticket enterprise bean and the data is returned in a TicketInfo object.
- The setCustomerHome() method creates a CustomerHome object by looking up the CustomerHome class.

- The setFlightHome() method creates a FlightHome object by looking up the FlightHome class.
- The setTicketHome() method creates a TicketHome object by looking up the TicketHome class.
- The updateCustomer() method updates a customer's personal information. The following tasks are performed:
    1. A find by primary key is executed based on the customer number using the Customer enterprise bean.
    2. The following methods are invoked in the Customer enterprise bean:
        - setCustomerFirstName() method
        - setCustomerMiddleInitial() method
        - setCustomerLastName() method
        - setCustomerAddress() method
        - setCustomerCity() method
        - setCustomerState() method
        - setCustomerZipCode() method
        - setCustomerCountry() method
        - setCustomerPhoneNumber() method
        - setCustomerInernetAddress() method
        - getData() method which stores the data in a CustomerInfo object
    3. The CustomerInfo object is returned.

## Customer bean

The Customer enterprise bean is an entity bean used to represent a flights customer. It supplies methods that accomplish tasks that are performed on customer data, for example:

- Viewing customer information
- Updating customer information

The Customer enterprise bean uses bean-managed persistence and is mapped to the FlightPerson form in the names.nsf Lotus Domino database. The layout for the FlightPerson form is shown in Table 10 in the ***Lotus Domino environment*** section.

The Customer enterprise bean contains one ejbFindBy method. This finder method is defined in the home interface. Table 1 contains the finder method.

**Table 1 Customer enterprise bean Finder Methods**

| Method Name | Select Statement | Description |
|---|---|---|
| ejbFindByPrimaryKey() | "SELECT (Form = \"FlightPerson\") & (PersonalID = " + customerNumber.trim() +")" | Finds customers based on customer number |

The getter and setter methods, and the values they return or set, are listed in Table 2.

**Table 2 Customer Getter and Setter Methods**

| Getter | Setter | Form Field Name | Data Type |
|---|---|---|---|
| getCustomerAddress() | setCustomerAddress() | FlightPerson: StreetAddress | String |
| getCustomerCity() | setCustomerCity() | FlightPerson: City | String |
| getCustomerCountry() | setCustomerCountry() | FlightPerson: Country | String |
| getCustomerFirstName() | setCustomerFirstName() | FlightPerson: FirstName | String |
| getCustomerInternetAddress() | setCustomerInternetAddress() | FlightPerson: InternetAddress | String |
| getCustomerLastName() | setCustomerLastName() | FlightPerson: LastName | String |
| getCustomerMiddleInitial() | setCustomerMiddleInitial() | FlightPerson: MiddleInitial | String |
| getCustomerNumber() | setCustomerNumber() | FlightPerson: PersonalID | String |
| getCustomerPhoneNumber() | setCustomerPhoneNumber() | FlightPerson: PhoneNumber | String |
| getCustomerState() | setCustomerState() | FlightPerson: State | String |
| getCustomerZipCode() | setCustomerZipCode() | FlightPerson: Zip | String |
| getData() | | CustomerInfo JavaBean containing the values within the enterprise bean | CustomerInfo |

The ejbLoad() method is used to retrieve the data from a specific FlightPerson document and place it in the bean properties.

The ejbStore() method is used to update the data in a specific FlightPerson document from the bean properties.

### CustomerInfo

CustomerInfo is a JavaBean used to store the customer information. It is passed to the appropriate JSP, which, in turn, uses it to retrieve the specific customer data.

The getter and setter methods, and the values that they return or set, are listed in Table 3.

**Table 3 CustomerInfo Getter and Setter Methods**

| Getter | Setter | Value | Data type |
|---|---|---|---|
| getCustomerAddress() | setCustomerAddress() | Customer Address | String |
| getCustomerCity() | setCustomerCity() | Customer City | String |
| getCustomerCountry() | setCustomerCountry() | Customer Country | String |

| | | | |
|---|---|---|---|
| getCustomerFirstName() | setCustomerFirstName() | Customer First Name | String |
| getCustomerInternetAddress() | setCustomerInternetAddress() | Customer Internet Address | String |
| getCustomerLastName() | setCustomerLastName() | Customer Last Name | String |
| getCustomerMiddleInitial() | setCustomerMiddleInitial() | Customer Middle Initial | String |
| getCustomerPhoneNumber() | setCustomerPhoneNumber() | Customer Phone Number | String |
| getCustomerState() | setCustomerState() | Customer State | String |
| getCustomerZipCode() | setCustomerZipCode() | Customer Zip Code | String |

The source code used within the Customer and CustomerInfo beans is shown in the ***Example: Customer bean*** section.

## Flight bean

The Flight enterprise bean is an entity bean used to represent a flight. It supplies methods that accomplish tasks that are performed on flight data, for example:

- Obtaining a list of available flights based on specific search criteria
- Obtaining specific flight information based on the flight by date value

The Flight enterprise bean uses bean-managed persistence and is mapped to the List Of Available Flights and the Scheduled Flights forms in the flights.nsf Lotus Domino database. The layout for the List of Available Flights form is shown in Table 2 and the Scheduled Flights form is shown in Table 3 in the ***Lotus Domino environment*** section.

The Flight enterprise bean contains several ejbFindBy methods. These finder methods are defined in the home interface. Table 4 contains the finder methods where the method name is preceded by ejbFindBy.

**Table 4 Flight Finder Methods**

| Method Name | Select Statement | Description |
|---|---|---|
| ArrivalCity() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\")"<br><br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival city |
| ArrivalCityArrivalDate() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\")"<br><br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival city and the arrival date |
| ArrivalCityArrivalDate DepartureCity() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br><br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival city, arrival date, and departure city |
| ArrivalCityArrivalDate DepartureCityDepartureDate() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br><br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival city, arrival date, departure city, and departure date |
| ArrivalCityArrivalDate DepartureCitySeatType() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival city, arrival date, departure city, departure date, and seat type |
| ArrivalCityArrivalDate DepartureDate() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\")"<br><br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival city, arrival date, and departure date |

| | | |
|---|---|---|
| ArrivalCityArrivalDate DepartureDateSeatType() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival city, arrival date, departure date, and seat type |
| ArrivalCityArrivalDate SeatType() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival city, arrival date, and seat type |
| ArrivalCityDepartureCity() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival city and departure city |
| ArrivalCityDepartureCity DepartureDate() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival city, departure city, and departure date |
| ArrivalCityDepartureCity DepartureDateSeatType() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival city, departure city, departure date, and seat type |
| ArrivalCityDepartureCity SeatType() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival city, departure city, and seat type |
| ArrivalCityDepartureDate() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\")"<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival city and departure date |
| ArrivalCityDepartureDate SeatType() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival city, departure date, and seat type |

| | | |
|---|---|---|
| ArrivalCitySeatType() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival city and seat type |
| ArrivalDate() | "SELECT (Form = \"Scheduled Flights\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival date |
| ArrivalDateDepartureCity() | "SELECT (Form = \"List Of Available Flights\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival date and departure city |
| ArrivalDateDepartureCity DepartureDate() | "SELECT (Form = \"List Of Available Flights\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival date, departure city, and departure date |
| ArrivalDateDepartureCity DepartureDateSeatType() | "SELECT (Form = \"List Of Available Flights\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival date, departure city, departure date, and seat type |
| ArrivalDateDepartureCity SeatType() | "SELECT (Form = \"List Of Available Flights\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival date, departure city, and seat type |
| ArrivalDateDepartureDate() | "SELECT (Form = \"Scheduled Flights\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the arrival date and departure date |
| ArrivalDateDepartureDate SeatType() | "SELECT (Form = \"List Of Available Flights\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival date, departure date, and seat type |
| ArrivalDateSeatType() | • Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the arrival date and seat type |
| DepartureCity() | "SELECT (Form = \"List Of Available Flights\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum +"\") & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the departure city |

| | | |
|---|---|---|
| DepartureCityDepartureDate() | "SELECT (Form = \"List Of Available Flights\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br><br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the departure city and departure date |
| DepartureCityDepartureDate SeatType() | "SELECT (Form = \"List Of Available Flights\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the departure city, departure date, and seat type |
| DepartureCitySeatType() | "SELECT (Form = \"List Of Available Flights\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the departure city and seat type |
| DepartureDate() | "SELECT (Form = \"Scheduled Flights\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) \| (FirstClassSeatsAvailable>0))" | Finds flights based on the departure date |
| DepartureDateSeatType() | • Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the departure date and seat type |
| PrimaryKey() | "SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightByDate = \""+ tempScheduledFlightByDate +"\")"<br><br>"SELECT ((Form = \"List Of Available Flights\")) & (AvailableFlightNumber = \"" + flightNum + "\")" | Finds flights based on the scheduled flight by date |
| SearchCriteria() | "SELECT (Form = \"List Of Available Flights\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")"<br>• Coach<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on all search criteria |
| SeatType() | • Coach<br>"SELECT (Form = \"Scheduled Flights\") & (CoachSeatsAvailable > 0) & (Status = \"New\")"<br>• First Class<br>"SELECT (Form = \"Scheduled Flights\") & (FirstClassSeatsAvailable > 0) & (Status = \"New\")" | Finds flights based on the seat type |

The ejbLoad() method is used to retrieve the data based on the scheduled flight by date from a Scheduled Flights document and the corresponding List Of Available Flights document and places it in the bean properties.

The getter and setter methods, and the values they return or set, are listed in Table 5.

**Table 5 Flight Getter and Setter Methods**

| Getter | Setter | Form Field Name | Data Type |
|---|---|---|---|
| getArrivalCityCode() | setArrivalCityCode() | List Of Available Flights: DisplayOnlyArrival | String |
| getArrivalDate() | setArrivalDate() | Scheduled Flights: ScheduledArrivalDate | String |
| getArrivalTime() | setArrivalTime() | List Of Available Flights: ArrivalTime | String |
| getCoachPrice() | setCoachPrice() | List Of Available Flights: CoachPrice | String |
| getCoachSeatsAvailable() | setCoachSeatsAvailable() | Scheduled Flights: CoachSeatsAvailable | String |
| getData() | | FlightInfo JavaBean containing the values within the enterprise bean bean | FlightInfo |
| getDepartureCityCode() | setDepartureCityCode() | List Of Available Flights: DisplayOnlyDeparture | String |
| getDepartureDate() | setDepartureDate() | Scheduled Flights: ScheduledDepartureDate | String |
| getDepartureTime() | setDepartureTime() | List Of Available Flights: DepartureTime | String |

| | | | |
|---|---|---|---|
| getFirstClassPrice() | setFirstClassPrice() | List Of Available Flights: FirstClassPrice | String |
| getFirstClassSeatsAvailable() | setFirstClassSeatsAvailable() | Scheduled Flights: FirstClassSeatsAvailable | String |
| getFood() | setFood() | List Of Available Flights: Food | String |
| getListAirplaneType() | setListAirplaneType() | List Of Available Flights: ListAirplaneType | String |
| getScheduledFlightByDate() | setScheduledFlightByDate() | Scheduled Flights: ScheduledFlightByDate | String |

### FlightInfo

FlightInfo is a JavaBean used to store the flight information. It is passed to the appropriate JSP, which, in turn, uses it to retrieve the specific flight data.

The getter and setter methods, and the values that they return or set, are listed in Table 6.

**Table 6 FlightInfo Getter and Setter Methods**

| Getter | Setter | Value | Data type |
|---|---|---|---|
| getArrivalCityCode() | setArrivalCityCode() | Flight Arrival City Code | String |
| getArrivalDate() | setArrivalDate() | Flight Arrival Date | String |
| getArrivalTime() | setArrivalTime() | Flight Arrival Time | String |
| getCityCode() | setCityCode() | Flight City Code | String |
| getCoachPrice() | setCoachPrice() | Flight Coach Price | String |
| getCoachSeatsAvailable() | setCoachSeatsAvailable() | Flight Coach Seats Available | String |
| getDepartureCityCode() | setDepartureCityCode() | Flight Departure City Code | String |
| getDepartureDate() | setDepartureDate() | Flight Departure Date | String |
| getDepartureTime() | setDepartureTime() | Flight Departure Time | String |
| getFirstClassPrice() | setFirstClassPrice() | Flight First Class Price | String |
| getFirstClassSeatsAvailable() | setFirstClassSeatsAvailable() | Flight First Class Seats Available | String |
| getFlightList() | setFlightList() | Flight List | Vector |
| getFood() | setFood() | Flight Food | String |
| getListAirplaneType() | setListAirplaneType() | Flight Airplane Type | String |
| getScheduledFlightByDate() | setScheduledFlightByDate() | Flight Scheduled Flight By Date | String |

## Ticket bean

The Ticket enterprise bean is an entity bean used to represent a ticket. It supplies methods that accomplish tasks that are performed on ticket data, for example:

- Creating a new ticket for a flight which has been booked by a customer
- Obtaining a list of tickets for a specified customer or itinerary number
- Marking the ticket as paid and saving the credit card information

The Ticket enterprise bean uses bean-managed persistence and is mapped to the Ticket Information and Credit Information forms in the flights.nsf Lotus Domino database. The layout for the Ticket Information form is shown in Table 5 and the Credit Information form is shown in Table 6 in the ***Lotus Domino environment*** section.

The Ticket enterprise bean contains several ejbFindBy methods. These finder methods are defined in the home interface. Table 7 contains the finder methods.

**Table 7 Ticket Finder Methods**

| Method Name | Select Statement | Description |
|---|---|---|
| ejbFindByCustomerNumber() | "SELECT ((Form=\"Ticket Information\")) & (TicketCustomerNumber = \"" + aCustomerNumber + "\")" | Finds tickets based on the customer number |
| ejbFindByInvoiceNumber() | "SELECT ((Form=\"Ticket Information\")) & (InvoiceNumber = \"" + aInvoiceNumber + "\")" | Finds tickets based on the invoice number |
| ejbFindByPrimaryKey() | "SELECT ((Form=\"Ticket Information\")) & (FlightByDate = \"" + tk.flightByDate + "\") & (SeatNumber = \"" + tk.seatNumber +"\")"<br><br>"SELECT ((Form=\"Credit Information\")) & (InvoiceNumber = \"" + invoiceNumber + "\")" | Finds tickets based on flight by date and seat number |

The ejbLoad() method is used to retrieve the data based on the flight by date and seat number from a Ticket Information document and the corresponding Credit Information document and places it in the bean properties.

The ejbStore() method is used to update the data in a specific Ticket Information document and the corresponding Credit Information document from the bean properties.

The getter and setter methods, and the values they return or set, are listed in Table 8.

**Table 8 Ticket Getter and Setter Methods**

| Getter | Setter | Form Field Name | Data Type |
|---|---|---|---|
| getCreditCardExpirationDate() | setCreditCardExpirationDate() | Credit Information: ExpirationDate | String |
| getCreditCardNumber() | setCreditCardNumber() | Credit Information: CreditCardNumber | String |
| getCreditCardType() | setCreditCardType() | Credit Information: CardType | String |

| | | | |
|---|---|---|---|
| getCustomerNumber() | setCustomerNumber() | Ticket Information: TicketCustomerNumber | String |
| getData() | | TicketInfo JavaBean containing the values within the enterprise bean bean | String |
| getFlightByDate() | setFlightByDate() | Ticket Information: FlightByDate | String |
| getInvoiceNumber() | setInvoiceNumber() | Ticket Information: InvoiceNumber | String |
| getPaidStatus() | setPaidStatus | Ticket Information: PaidStatus | String |
| getPassengerAddress() | setPassengerAddress() | Ticket Information: PassengerStreet | String |
| getPassengerCity() | setPassengerCity() | Ticket Information: PassengerCity | String |
| getPassengerCountry() | setPassengerCountry() | Ticket Information: PassengerCountry | String |
| getPassengerFirstName() | setPassengerFirstName() | Ticket Information: PassengerFirstName | String |
| getPassengerLastName() | setPassengerLastName() | Ticket Information: PassengerLastName | String |
| getPassengerMiddleInitial() | setPassengerMiddleInitial() | Ticket Information: PassengerMiddleInitial | String |
| getPassengerPhoneNumber() | setPassengerPhoneNumber() | Ticket Information: PassengerPhone | String |
| getPassengerState() | setPassengerState() | Ticket Information: PassengerState | String |
| getPassengerZipCode() | setPassengerZipCode() | Ticket Information: PassengerZip | String |
| getSeatNumber() | setSeatNumber() | Ticket Information: SeatNumber | String |
| getTicketClass() | setTicketClass() | Ticket Information: TicketClass | String |
| getTicketPrice() | setTicketPrice() | Ticket Information: TicketPrice | BigDecimal |
| getTicketStatus() | setTicketStatus() | Ticket Information: TicketStatus | String |
| payTicket() | | Ticket Information:<br>PaidStatus and sets Credit Information | String |

### TicketInfo

TicketInfo is a JavaBean used to store the ticket information. It is passed to the appropriate JSP, which, in turn, uses it to retrieve the specific ticket data.

The getter and setter methods, and the values that they return or set, are listed in Table 9.

**Table 9 TicketInfo Getter and Setter Methods**

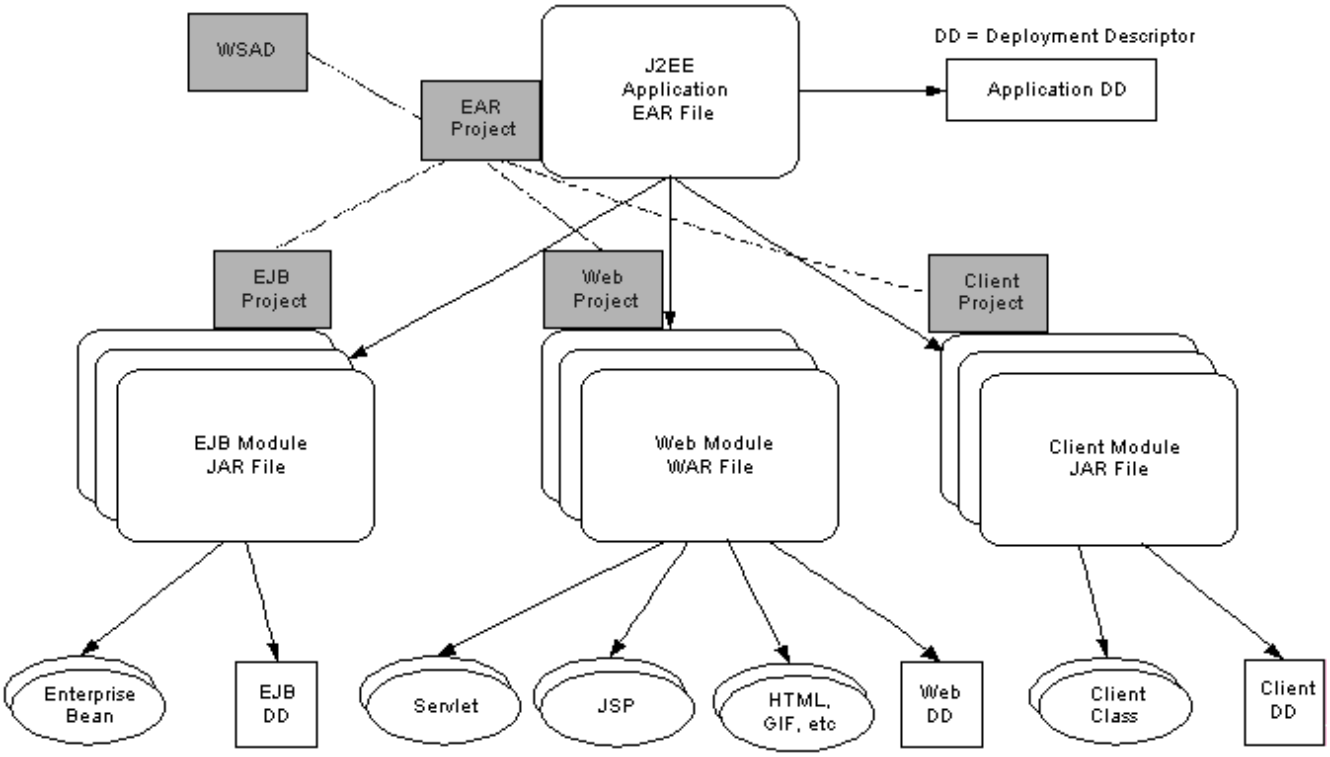
| Getter | Setter | Value | Data type |
|---|---|---|---|
| getCreditCardExpirationDate() | setCreditCardExpirationDate() | Credit card expiration date | String |
| getCreditCardNumber() | setCreditCardNumber() | Credit card number | String |
| getCreditCardType() | setCreditCardType() | Credit card type | String |
| getCustomerNumber() | setCustomerNumber() | Customer number | String |
| getFlightByDate() | setFlightByDate() | Flight by date | String |
| getInvoiceNumber() | setInvoiceNumber() | Invoice number | String |
| getPaidStatus() | setPaidStatus() | Paid status | String |
| getPassengerAddress() | setPassengerAddress() | Passenger address | String |
| getPassengerCity() | setPassengerCity() | Passenger city | String |
| getPassengerCountry() | setPassengerCountry() | Passenger country | String |
| getPassengerFirstName() | setPassengerFirstName() | Passenger first name | String |
| getPassengerLastName() | setPassengerLastName() | Passenger last name | String |
| getPassengerMiddleInitial() | setPassengerMiddleInitial() | Passenger middle initial | String |
| getPassengerPhoneNumber() | setPassengerPhoneNumber() | Passenger phone number | String |
| getPassengerState() | setPassengerState() | Passenger state | String |
| getPassengerZipCode() | setPassengerZipCode() | Passenger zip code | String |
| getSeatNumber() | setSeatNumber() | Seat number | String |
| getTicketClass() | setTicketClass() | Ticket class | String |
| getTicketList() | setTicketList() | Ticket list | Vector |
| getTicketPrice() | setTicketPrice() | Ticket price | String |
| getTicketStatus() | setTicketStatus() | Ticket status | String |
| getTotalCost() | | Total cost of all tickets within the ticket list vector | String |
| getUniqueInvoiceNumbers() | | Unique invoice numbers of all tickets within the ticket list vector | Vector |
| toString() | | String representation of the object | String |

## Installation of Enterprise Application

WebSphere Studio Application Developer combines the functionality that was found in Visual Age for Java(TM) and WebSphere Studio. However, many new features were added. It supports perspectives, which allow you to work with your applications from different views. For example, the J2EE perspective allows you to work in an environment customized for building J2EE compliant applications.

Application Developer allows you to export your applications directly into J2EE compliant formats, such as enterprise archive file (EAR) and Web archive file (WAR). You can install these files as enterprise applications in WebSphere Application Server 4.0 without using the WebSphere Application Assembly Tool (AAT). Figure 2 shows the J2EE hierarchy and the matching support in WebSphere Studio Application Developer.

**Figure 2 J2EE Architecture**




A J2EE application is stored in an Enterprise Archive (EAR) file that contains enterprise bean modules (stored in an enterprise bean JAR file), Web modules (stored in Web Archives (WAR) files), and client modules (stored in a JAR file). A WAR file contains all the components of a Web application: servlets, JSPs, HTML files, images, and so on. Each of the modules contains a deployment descriptor. For example, a WAR file contains a web.xml file.

The J2EE hierarchy is matched by projects in WSAD. An EAR project contains references to enterprise bean, Web, and Client projects.

Within this application, there is a FlightsEAR EAR file which contains a FlightsEJBModule and a FlightsWebModule Web module. The FlightsEJBModule contains all of the enterprise beans (CustomerFlight, Customer, Flight, and Ticket). The FlightWebModule contains all of the Web components (FlightsServlet, HTML, JSP, and image files).

To generate the flights Enterprise Application file:

1. Select File -> Export from the Application Developer main screen
2. The Export window will appear at which point choose EAR file
3. In the next window, select the FlightsEAR resource from the drop down list for the "What resource do you want to export?" field.
4. Enter the location SystemName:\QIBM\UserData\WebASAdv4\instanceName\installableApps\flight.ear for the "Where do you want to export resources to?" field. Click Finish.
5. Once the flights Enterprise Application is exported you can see the file in the folder.

To install the flights Enterprise Application:

1. Open a command prompt window to start the Administrative Console. Wait until you see the message Console Ready.
2. In the Console, select the wizard icon and click Install Enterprise Application. The Specifying the Application Module window displays. Make sure that the Install Application radio button is selected. Click the Browse button next to Path to locate the flight.ear in the \QIBM\UserData\WebASAdv4\instanceName\installableApps directory.
3. After clicking on Next, you will get the following message: "This application contains method permissions. Do you wish to deny all unprotected methods?" Answer Yes.
4. On the Mapping User Roles window, press the Select button and check "All Authenticated Users" then press OK.
5. Keep clicking on Next until you see the "Selecting Application Servers" window. At this window, select all the modules in the Module box and press the Select Server button. Choose the default server and press OK.
6. Click Next and then Finish to install the application, when the Regenerate the application dialog displays, Click No.

Now that the flights Enterprise Application is installed on WebSphere Application Server Version 4.0 Advanced Edition, we need to stop the server. Before we restart it, we need to copy the NCSOW.jar to the QIBM/UserData/WebASAdv4/InstanceName/lib/ext directory so that it is picked up in the classpath when the server is restarted. Restart the server to make the new application ready.

Finally the flights Enterprise Application is ready for use. To use the application, open a browser and enter the following URL: http://systemName:port/webapp/Flights/index.html

# WebSphere(R) Application Server environment key findings

Following is a list of key findings that we uncovered while creating and using the flights scenario WebSphere Application Server environment.

- To set the Session Timeout value in a stateful session bean to a larger value, perform the following steps in the WebSphere Application Server Application Assembly Tool (AAT):
  1. Open your enterprise bean JAR or EAR in the AAT
  2. Drill down to your Session enterprise bean and click on it.
  3. Click on the IBM(R) Extensions tab in the right-hand pane
  4. Set the Timeout property
  5. Save your JAR or EAR
  6. Exit the AAT

- WebSphere Application Server Application Assembly Tool (AAT) can be used to generate code for an Enterprise Application. If you encounter problems generating code for an Enterprise Application during installation, the WebSphere Application Server administration console does not provide much information on why installation failed. In this case, you can use AAT to generate the deployed code for the same EAR file and it will provide you with more information as to why it is failing. Both the WebSphere Application Server administration console and AAT use the same underlying code to generate the code. There is also a Verify option that you can run on the EAR file which will give you additional information.

- To accommodate the needs of the JSPs, a couple of the Lotus(R) Domino(TM) documents had to have fields added to them. For the flightSearch.jsp, the Scheduled Flights form needed to have the following fields added: Departure Date and Arrival Date. For the passengerInformation.jsp, the Ticket Information form needed to have passenger information fields added.

- When using the Lotus Domino JDBC driver, the names of the Lotus Domino documents and columns are case sensitive within the SQL statements.

- In the flights application, we used the Lotus Domino Java(TM) APIs to connect to a Lotus Domino database. Originally we were going to use the Lotus Domino JDBC driver; however, in WebSphere Application Server, you can not create a datasource that uses the Lotus Domino JDBC driver on the iSeries(TM). Since the iSeries has not yet ported the Lotus Domino JDBC driver, we also could not implement our own connection pooling. A Lotus Domino JDBC driver for iSeries is scheduled to be available in a future release.

- The following hints may help if you are experiencing problems connecting to a Lotus Domino server remotely from a WebSphere Java Application:

  - Make sure DIIOP is set up on the Lotus Domino server. If DIIOP is not set up, you will receive a message that the remote host refused the connection.

  - Make sure to specify the port number of the Lotus Domino HTTP server when trying to obtain a NotesFactory session. For example:
    Session session = NotesFactory.createSession("systemName.domainName:portNumber", "user ID", "password").
    This will allow the CORBA request to get to the correct server. If the port number is not specified, you will receive a NotesException.

  - You may receive a NotesException of 4377: Server must be on same host as session when performing the getDatabase statement from a Java Application using the following lines of code:
    Session session = NotesFactory.createSession(notesServer, notesUser, password);
    ndbContent = session.getDatabase(notesServer, notesDatabase);

    There are two solutions:
    1. On the getDatabase, send only the server name not the server name along with the port number.
    2. Pass in "" as the notesServer on the getDatabase call which will force the use of the session just created:
       ndbContent = session.getDatabase("", notesDatabase);

  - Make sure that the CLASSPATH contains the NCSOW.jar file obtained from the version of Lotus Domino that you are using. This is the WebSphere version of the NCSO.jar file. The NCSO.jar is unusable because the IIOP levels will clash and you will not be able to create a session.

  - To use the Domino Java APIs to connect from a WebSphere Java Application to a Domino database via IIOP, the application must import the lotus.domino.* package.

  - To obtain additional information on NotesExceptions, add the System.out.println in the code below. This allows you to print out static variables that explain the error better:

```
catch(lotus.domino.NotesException ne)
{
System.out.println(ne.text + " " + ne.id);
ne.printStackTrace();
}
```

- In a Java application using Lotus Domino Java APIs, when writing a Select statement to search on a date, you need to specify the date you are searching for as a constant (ie the [ ] around the value). For example: SELECT (Form = "Scheduled Flights") & (ScheduledDepartureDate = [10/31/2001]).

- When coding finder methods in bean-managed persistence entity beans, the finder is implemented by the ejbFindBy*xxxx*() method. Finder methods in the home interface will have the name findBy*xxxx*(). The container implements the findBy*xxxx*() essentially as a wrapper around ejbFindBy*xxxx*() so the actual finder code needs to be written.

- To avoid receiving a CSITransactionRolledbackException when calling a bean-managed persistence entity bean from a session bean, the session bean needs to be deployed with a transaction setting of TX_SUPPORTS.

  A org.omg.CORBA.INV_OBJREF exception happens when the client code is trying to use an object that the Lotus Domino server does not know about any more. This could happen if the client session stays idle longer than the amount of time allowed by the session timeout parameter in the server record or if someone forces a drop of all the sessions from the server console. Leaving a session idle for a long time is not considered to be a good thing so DIIOP will go out and terminate these sessions. The client is responsible for handling the situation.

  If the client does not handle the error, the Lotus Domino Session will time out and render the session object obsolete and the following error messages will be displayed.

  Here is the error message from the WebSphere Application Server:

```
org.omg.CORBA.INV_OBJREF: minor code: 1229062208 completed: No
java/lang/Throwable.<init>(Ljava/lang/String;)V+4 (Throwable.java:94)
org/omg/CORBA/INV_OBJREF.<init>(Ljava/lang/String;ILorg/omg/CORBA/CompletionStatus;)V+1 (INV_OBJREF.java:72)
org/omg/CORBA/INV_OBJREF.<init>(Ljava/lang/String;)V+6 (INV_OBJREF.java:48)
com/ibm/CORBA/iiop/ReplyMessage.getSystemException()Lorg/omg/CORBA/SystemException;+119 (ReplyMessage.java:181)
com/ibm/rmi/iiop/ClientResponseImpl.getSystemException()Lorg/omg/CORBA/SystemException;+11 (ClientResponseImpl.java:89)
com/ibm/CORBA/iiop/ClientDelegate.invoke(Lorg/omg/CORBA/Object;Lorg/omg/CORBA/portable/OutputStream;)Lorg/omg/CORBA/portable/InputStream;+235 (ClientDelegate.java:439)
org/omg/CORBA/portable/ObjectImpl._invoke(Lorg/omg/CORBA/portable/OutputStream;)Lorg/omg/CORBA/portable/InputStream;+4 (ObjectImpl.java:251)
```

lotus/domino/corba/_IDatabaseStub.search(Ljava/lang/String;Llotus/domino/corba/IDateTime;I)Llotus/domino/corba/DCData;+0 (_IDatabaseStub.java:0)
lotus/domino/cso/Database.search(Ljava/lang/String;Llotus/domino/DateTime;I)Llotus/domino/DocumentCollection;+0 (Database.java:1478)
lotus/domino/cso/Database.search(Ljava/lang/String;)Llotus/domino/DocumentCollection;+0 (Database.java:1452)
com/flights/ejb/session/CustomerFlightBean.getAllCityCodes()Lcom/flights/FlightInfo;+0 (CustomerFlightBean.java:110)
com/flights/ejb/session/EJSRemoteCustomerFlight.getAllCityCodes()Lcom/flights/FlightInfo;+0 (EJSRemoteCustomerFlight.java:31)
com/flights/ejb/session/_EJSRemoteCustomerFlight_Tie._invoke(Ljava/lang/String;Lorg/omg/CORBA/portable/InputStream;Lorg/omg/CORBA/portable/ResponseHandler;)Lorg/omg/CORBA/portable/OutputStream;+0 (_EJSRemoteCustomerFlight_Tie.java:82)
com/ibm/CORBA/iiop/ExtendedServerDelegate.dispatch(Lcom/ibm/rmi/ServerRequest;)Lcom/ibm/rmi/ServerResponse;+224 (ExtendedServerDelegate.java:506)
com/ibm/CORBA/iiop/ORB.process(Lcom/ibm/rmi/ServerRequest;)Lcom/ibm/rmi/ServerResponse;+20 (ORB.java:2282)
com/ibm/CORBA/iiop/WorkerThread.run()V+89 (WorkerThread.java:195)
com/ibm/ejs/oa/pool/ThreadPool$PooledThread.run()V+67 (ThreadPool.java:641)

Here is the error message from the Domino server:

DIIOP SYSTEM EXCEPTION: INV_OBJREF, minor code 49420040, SOMDERROR_BadObjref [somd_refdata_to_obj(CORBA::ReferenceData*):1091]

You can try setting the timeout of IIOP to a higher value or you can try to catch the error and re-establish the session.

The following are ideas we tried to use to determine if a Lotus Domino session object is open from a Java client:

- Catch the exception from within the enterprise bean and handle it there. This idea will not work because of the following:.

  From the Enterprise JavaBean 1.1 specification, section 12.3.4 Exceptions and transactions:

  "If an instance has thrown an unchecked exception while executing in a client's transaction context, the container must mark the transaction for rollback and throw javax.jts.TransactionRolledbackException to the client."

  "If the container decides for any reason to mark a transaction for rollback, it should throw the javax.jts.TransactionRolledbackException to the client. The javax.jts.TransactionRolledbackException is a subclass of the java.rmi.RemoteException, and it informs the client that any attempted recovery of the exception within the transaction would be fruitless since the transaction cannot commit."

  To summarize, an unchecked exception occurring in an enterprise bean will always cause the in-flight transaction to be rolled back even if you are explicitly handling it. Basically, an unchecked exception is any exception that is not derived from java.lang.Exception. A CORBA.INV_OBJREF exception is not derived from java.lang.Exception; hence, it can be classified as an unchecked exception. Thus, the container will always throw a TransactionRolledbackException when this exception is thrown.

- Use Session.isValid() which will be added to a future version of the Lotus Domino Java APIs. This is the best way to determine the state of a session, but it is not yet available.

  Feature: Session.isValid()
  This is part of the DIIOP connection pooling feature.

  Purpose:
  The purpose of this Java only method is to determine if a Session object that had been created is still valid. For the remoted API, it determines if the DIIOP server task still considers this session valid and therefore this method may perform a network operation. For this reason, this method should not be used in a tight loop.
  For the local API, it also determines if a session is still valid.

  Signature:
  boolean isValid();

  NOTE: This method does not throw any Exceptions.

  Usage:

  Here's an example of how to use the method in a servlet worker type of thread.

```
class WorkerThread extends Thread {
...
public void run()
{
while (WaitForWork()) {
if ( ! session.isValid() ) {
// need to create new session
}
// do the work
}
}
}
```

- Modify your getConnection code to always get a new session to Lotus Domino. This will avoid the CORBA.INV_OBJREF exception all together. This idea works although it provides extra overhead to the Java application.

- In the server document on the IIOP tab, the maximum number of threads that the administrator can specify is not restricted. In a future release of Lotus Domino, this setting is no longer used by DIIOP. It will be left in the server document only for backwards support. The minimum timeout value for a session in the Lotus Domino server document is five minutes.

- To specify classpath information for objects that reside in QIBM without using AAT, in WebSphere Studio Application Developer, you will want to create a MANIFEST.MF file in the following locations:

  For an enterprise bean module, you want create or use the existing manifest file in the following location : ejbModule -> META-INF -> MANIFEST.MF

  For a Web module, you want to create or use the existing manifest file in the following location: webApplication -> META-INF -> MANIFEST.MF

  Here is an example of what the classpath would look like in a Manifest.mf file:

  Manifest-Version: 1.0
  Class-Path: /qibm/userdata/webasadv4/flight4/installedapps/flightsear.ear/flightsejbmodule.jar

- After importing the published versions of the flights JSPs into WebSphere Studio Application Developer, they were each edited using Page Designer. After editing, the EAR was exported to the installable directory and then each JSP was exported to a central team location. The JSPs that had updates made to them would not export to the team location. We also found that these JSPs could not be copied or deleted within WebSphere Studio Application Developer. The error received was that the resource is out of sync with the file system. The JSPs that were opened but did not have changes made were okay. The Web Module was then refreshed from local. This allowed the JSPs to be copied, deleted, and exported.

- Table 1 lists the keyboard shortcuts you can use in WebSphere Studio Application Developer's Java editor:

**Table 1 Keyboard shortcuts**

| Description | Key Sequence |
|---|---|
| Import | Ctrl+Shift+M |
| Go to line number | Ctrl+L |
| Indent the highlighted text | Ctrl+I |
| Find/replace | Ctrl+F |
| Copy | Ctrl+C |
| Cut | Ctrl+X |
| Undo | Ctrl+Z |
| Select all | Ctrl+A |
| Go to the next error | Ctrl+E |
| Brings up Java search with the selected item in the search table | Ctrl+H |
| Brings up coding/content assistant. After you make your selection, Javadoc appears in hover Help | Ctrl+Space |
| Executes an incremental build of a project in the navigation view | Ctrl+B |
| Hold Ctrl key down and drag-and-drop resource to copy the resource between different Workbench windows | Ctrl+Drag-and-Drop |

# References


- Developing iSeries J2EE Applications for WebSphere 4.0, IBM Redbook SG24-6559-00
- WebSphere Studio Application Developer Programming Guide, IBM Redbook SG24-6585-00
- Tips for Working with Lotus Domino Objects
  http://www.advisor.com/Articles.nsf/aidp/BALAB03
- WebSphere Studio Application Developer Migration Guide
  http://www7b.boulder.ibm.com/wsdd/library/techarticles/0110_wsad_mig/migration_ga.html

# Example: Customer bean

The following examples illustrate the coding of the Customer entity bean. The Customer bean uses bean-managed persistence and is mapped to the FlightPerson form in the names.nsf Lotus(R) Domino(TM) database. It uses Lotus Domino APIs to access the Lotus Domino database.

The source code for the CustomerKey is shown in Example 1.

**Example 1:** CustomerKey source code

```
package com.flights.ejb.bmp;

/**
* This is a Primary Key Class for the Entity Bean
**/
public class CustomerKey implements java.io.Serializable {
public String primaryKey;
final static long serialVersionUID = 3206093459760846163L;

/**
* CustomerKey() constructor
*/
public CustomerKey() {
}
/**
* CustomerKey(String key) constructor
*/
public CustomerKey(String key) {
primaryKey = key;
}
/**
* equals method
* - user must provide a proper implementation for the equal method. The generated
* method assumes the key is a String object.
*/
public boolean equals (Object o) {
if (o instanceof CustomerKey)
return primaryKey.equals(((CustomerKey)o).primaryKey);
else
return false;
}
/**
* hashode method
* - user must provide a proper implementation for the hashCode method. The generated
* method assumes the key is a String object.
*/
public int hashCode () {
return primaryKey.hashCode();
}
}
```

The source code for the CustomerHome inteface is shown in Example 2.

**Example 2:** CustomerHome source code

```
package com.flights.ejb.bmp;

/**
```

```
* This is a Home interface for the Entity Bean
*/
public interface CustomerHome extends javax.ejb.EJBHome {

/**
* create method for a BMP entity bean
* @return com.flights.ejb.bmp.Customer
* @param primaryKey com.flights.ejb.bmp.CustomerKey
* @exception javax.ejb.CreateException The exception description.
* @exception java.rmi.RemoteException The exception description.
*/
com.flights.ejb.bmp.Customer create(com.flights.ejb.bmp.CustomerKey primaryKey) throws
javax.ejb.CreateException, java.rmi.RemoteException;
/**
* findByPrimaryKey method comment
* @return com.flights.ejb.bmp.Customer
* @param key com.flights.ejb.bmp.CustomerKey
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.FinderException The exception description.
*/
com.flights.ejb.bmp.Customer findByPrimaryKey(com.flights.ejb.bmp.CustomerKey key) throws
java.rmi.RemoteException, javax.ejb.FinderException;
}
```

The source code for the Customer remote interface is shown in Example 3.

**Example 3:** Customer remote interface source code

```
package com.flights.ejb.bmp;

/**
* This is the enterprise bean Remote Interface for the Customer bean
*/
public interface Customer extends javax.ejb.EJBObject {

/**
* Returns the address for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerAddress() throws java.rmi.RemoteException;
/**
* Returns the city for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerCity() throws java.rmi.RemoteException;
/**
* Returns the country for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerCountry() throws java.rmi.RemoteException;
/**
* Returns the first name of the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerFirstName() throws java.rmi.RemoteException;
/**
```

```
* Returns the internet address for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerInternetAddress() throws java.rmi.RemoteException;
/**
* Returns the last name of the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerLastName() throws java.rmi.RemoteException;
/**
* Returns the middle initial of the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerMiddleInitial() throws java.rmi.RemoteException;
/**
* Returns the customer number.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerNumber() throws java.rmi.RemoteException;
/**
* Returns the customer phone number.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerPhoneNumber() throws java.rmi.RemoteException;
/**
* Returns the state for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerState() throws java.rmi.RemoteException;
/**
* Returns the zip code for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerZipCode() throws java.rmi.RemoteException;
/**
* Returns the values within the bean as data stored within a CustomerInfo JavaBean.
* @return com.flights.CustomerInfo
* @exception String The exception description.
*/
com.flights.CustomerInfo getData() throws java.rmi.RemoteException;
/**
* Sets the address for the customer.
* @return void
* @param newCustomerAddress java.lang.String
* @exception String The exception description.
*/
void setCustomerAddress(java.lang.String newCustomerAddress) throws java.rmi.RemoteException;
/**
* Sets the city for the customer.
* @return void
* @param newCustomerCity java.lang.String
* @exception String The exception description.
*/
```

```
void setCustomerCity(java.lang.String newCustomerCity) throws java.rmi.RemoteException;
/**
* Sets the country for the customer.
* @return void
* @param newCustomerCountry java.lang.String
* @exception String The exception description.
*/
void setCustomerCountry(java.lang.String newCustomerCountry) throws java.rmi.RemoteException;
/**
* Sets the first name of the customer.
* @return void
* @param newCustomerFirstName java.lang.String
* @exception String The exception description.
*/
void setCustomerFirstName(java.lang.String newCustomerFirstName) throws java.rmi.RemoteException;
/**
* Sets the internet address for the customer.
* @return void
* @param newCustomerInternetAddress java.lang.String
* @exception String The exception description.
*/
void setCustomerInternetAddress(java.lang.String newCustomerInternetAddress) throws java.rmi.RemoteException;
/**
* Sets the last name of the customer.
* @return void
* @param newCustomerLastName java.lang.String
* @exception String The exception description.
*/
void setCustomerLastName(java.lang.String newCustomerLastName) throws java.rmi.RemoteException;
/**
* Sets the middle initial of the customer.
* @return void
* @param newCustomerMiddleInitial java.lang.String
* @exception String The exception description.
*/
void setCustomerMiddleInitial(java.lang.String newCustomerMiddleInitial) throws java.rmi.RemoteException;
/**
* Sets the customer number.
* @return void
* @param newCustomerNumber java.lang.String
* @exception String The exception description.
*/
void setCustomerNumber(java.lang.String newCustomerNumber) throws java.rmi.RemoteException;
/**
* Sets the phone number for the customer.
* @return void
* @param newCustomerPhoneNumber java.lang.String
* @exception String The exception description.
*/
void setCustomerPhoneNumber(java.lang.String newCustomerPhoneNumber) throws java.rmi.RemoteException;
/**
* Sets the state for the customer.
* @return void
* @param newCustomerState java.lang.String
* @exception String The exception description.
*/
void setCustomerState(java.lang.String newCustomerState) throws java.rmi.RemoteException;
/**
* Sets the zip code for the customer.
* @return void
```

```
* @param newCustomerZipCode java.lang.String
* @exception String The exception description.
*/
void setCustomerZipCode(java.lang.String newCustomerZipCode) throws java.rmi.RemoteException;
}
```

The source code for the Customer enterprise bean is shown in Example 4.

**Example 4:** Customer enterprise bean source code

```
package com.flights.ejb.bmp;

import java.rmi.RemoteException;
import java.security.Identity;
import java.util.Properties;
import javax.ejb.*;
import lotus.domino.*;
import javax.naming.*;
/**
* This is an Entity Bean class with BMP fields
*/
public class CustomerBean implements EntityBean {
private javax.ejb.EntityContext entityContext = null;
private final static long serialVersionUID = 3206093459760846163L;

private java.lang.String customerAddress;
private java.lang.String customerCity;
private java.lang.String customerCountry;
private java.lang.String customerFirstName;
private java.lang.String customerInternetAddress;
private java.lang.String customerLastName;
private java.lang.String customerMiddleInitial;
private java.lang.String customerPhoneNumber;
private java.lang.String customerState;
private java.lang.String customerZipCode;
private java.lang.String customerNumber;
private transient Database ndbContent = null;
/**
* ejbActivate method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbActivate() throws java.rmi.RemoteException {}
/**
* ejbCreate method for a BMP entity bean
* @return com.flights.ejb.bmp.CustomerKey
* @exception javax.ejb.CreateException The exception description.
* @exception java.rmi.RemoteException The exception description.
*/
public com.flights.ejb.bmp.CustomerKey ejbCreate() throws javax.ejb.CreateException, java.rmi.RemoteException {
return null;
}
/**
* ejbCreate method for a BMP entity bean
* @return com.flights.ejb.bmp.CustomerKey
* @param key com.flights.ejb.bmp.CustomerKey
* @exception javax.ejb.CreateException The exception description.
* @exception java.rmi.RemoteException The exception description.
*/
public com.flights.ejb.bmp.CustomerKey ejbCreate(com.flights.ejb.bmp.CustomerKey key) throws
javax.ejb.CreateException, java.rmi.RemoteException {
```

```
return null;
}
/**
* ejbFindByPrimaryKey method comment
* @return com.flights.ejb.bmp.CustomerKey
* @param primaryKey com.flights.ejb.bmp.CustomerKey
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.FinderException The exception description.
*/
public com.flights.ejb.bmp.CustomerKey ejbFindByPrimaryKey(com.flights.ejb.bmp.CustomerKey primaryKey)
throws java.rmi.RemoteException, javax.ejb.FinderException {
refresh(primaryKey);
return primaryKey;
}
/**
* Used to refresh the enterprise bean from the persistent storage.
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbLoad() throws java.rmi.RemoteException {

System.out.println("Customer.ejbLoad()");
try
{
refresh((CustomerKey) entityContext.getPrimaryKey());
}
catch (FinderException fe)
{
throw new RemoteException(fe.getMessage());
}
}
/**
* ejbPassivate method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbPassivate() throws java.rmi.RemoteException {}
/**
* ejbPostCreate method for a BMP entity bean
* @param key com.flights.ejb.bmp.CustomerKey
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbPostCreate(com.flights.ejb.bmp.CustomerKey key) throws java.rmi.RemoteException {}
/**
* ejbRemove method comment -- currently not implemented
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.RemoveException The exception description.
*/
public void ejbRemove() throws java.rmi.RemoteException, javax.ejb.RemoveException {}
/**
* ejbStore method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbStore() throws java.rmi.RemoteException {

System.out.println("Customer.ejbStore() ");

DocumentCollection dclResult = null;
Document docResult = null;

try
```

```
{
ndbContent = getConnection();

// Search for document with specified key values
System.out.println("Searching for document: " + customerNumber);
dclResult = ndbContent.search("SELECT (Form = \"FlightPerson\") & (PersonalID = "+ customerNumber.trim()
+")");
docResult = dclResult.getFirstDocument();

if (docResult != null)
{
// Update document to contain new values
System.out.println("Should be updating customer info");

docResult.replaceItemValue("FirstName", customerFirstName);
docResult.replaceItemValue("MiddleInitial", customerMiddleInitial);
docResult.replaceItemValue("LastName", customerLastName);
docResult.replaceItemValue("StreetAddress", customerAddress);
docResult.replaceItemValue("City", customerCity);
docResult.replaceItemValue("State", customerState);
docResult.replaceItemValue("Zip", customerZipCode);
docResult.replaceItemValue("Country", customerCountry);
docResult.replaceItemValue("PhoneNumber", customerPhoneNumber);
docResult.replaceItemValue("InternetAddress", customerInternetAddress);

docResult.save();
}
else
throw new FinderException("Customer EJB Store: CustomerBean (" + customerNumber + ") not found");

System.out.println("After update");

}
catch(lotus.domino.NotesException ne)
{
System.out.println(ne.text + " " + ne.id);
ne.printStackTrace();
throw new RemoteException(ne.toString());
}
catch(Exception e)
{
e.printStackTrace();
throw new RemoteException(e.toString());
}

finally
{
try
{
System.out.println("Recycle objects");
if (dclResult != null)
dclResult.recycle();
if (docResult != null)
docResult.recycle();
}
catch(Exception ex)
{
throw new RemoteException(ex.toString());
}
}
```

```
}
/**
* Used to return a connection to the Lotus Domino database using Lotus Domino APIs .
* Creation date: (10/19/2001 3:14:11 PM)
* @return javax.sql.DataSource
* @exception java.sql.SQLException The exception description.
*/
private Database getConnection() throws java.rmi.RemoteException, java.sql.SQLException {

if (ndbContent == null) {
Properties properties = getEntityContext().getEnvironment();
String providerURL = properties.getProperty("provider_url");
String notesServer = properties.getProperty("notesServer");
String notesUser = properties.getProperty("notesUser");
String password = properties.getProperty("password");
String notesDatabase = properties.getProperty("notesDB");

InitialContext ctx = null;
Properties prop = new Properties();

try {
prop.put(Context.PROVIDER_URL, providerURL);
prop.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
ctx = new InitialContext(prop);

System.out.println("creating notes session using current creds");
Session session = NotesFactory.createSession(notesServer, null);
System.out.println("username: " + session.getUserName());

System.out.println("got session - getting DB");
ndbContent = session.getDatabase("", notesDatabase);

System.out.println("got database");

if (!ndbContent.isOpen()) ndbContent.open();
}
catch(lotus.domino.NotesException ne){
System.out.println(ne.text + " " + ne.id);
ne.printStackTrace();
throw new RemoteException(ne.toString());
}
catch (Exception e) {
System.out.println("an error occurred");
e.printStackTrace();
throw new RemoteException(e.toString());
}
}

return ndbContent;
}
/**
* Returns address for customer.
* Creation date: (04/02/02 2:37:06 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerAddress() {
return customerAddress;
}
/**
```

```
* Returns city for customer.
* Creation date: (04/02/02 2:37:28 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerCity() {
return customerCity;
}
/**
* Returns country for customer.
* Creation date: (04/02/02 2:37:46 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerCountry() {
return customerCountry;
}
/**
* Returns first name of customer.
* Creation date: (04/02/02 2:38:05 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerFirstName() {
return customerFirstName;
}
/**
* Returns internet address of customer.
* Creation date: (04/02/02 2:38:24 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerInternetAddress() {
return customerInternetAddress;
}
/**
* Returns last name of customer.
* Creation date: (04/02/02 2:38:39 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerLastName() {
return customerLastName;
}
/**
* Returns middle initial of customer.
* Creation date: (04/02/02 2:38:55 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerMiddleInitial() {
return customerMiddleInitial;
}
/**
* Returns customer number.
* Creation date: (04/02/02 2:40:12 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerNumber() {
return customerNumber;
}
/**
* Returns phone number for customer.
* Creation date: (04/02/02 2:39:10 PM)
* @return java.lang.String
*/
```

```
public java.lang.String getCustomerPhoneNumber() {
return customerPhoneNumber;
}
/**
* Returns state for customer.
* Creation date: (04/02/02 2:39:22 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerState() {
return customerState;
}
/**
* Returns zip code for customer.
* Creation date: (04/02/02 2:39:40 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerZipCode() {
return customerZipCode;
}
/**
* Returns the values within the enterprise bean as data stored within a CustomerInfo JavaBean.
* Creation date: (03/26/02 7:20:12 AM)
*/
public com.flights.CustomerInfo getData() {

com.flights.CustomerInfo customerInfo = new com.flights.CustomerInfo();

customerInfo.setCustomerFirstName(customerFirstName);

System.out.println("getData(): Here's the customer first name: " + customerFirstName);

customerInfo.setCustomerMiddleInitial(customerMiddleInitial);

System.out.println("getData(): Here's the customer middle initial: " + customerMiddleInitial);
customerInfo.setCustomerLastName(customerLastName);
customerInfo.setCustomerAddress(customerAddress);
customerInfo.setCustomerInternetAddress(customerInternetAddress);

System.out.println("getData(): Here's the customer internet address: " + customerInternetAddress);
customerInfo.setCustomerCity(customerCity);
customerInfo.setCustomerState(customerState);
customerInfo.setCustomerZipCode(customerZipCode);
customerInfo.setCustomerCountry(customerCountry);
customerInfo.setCustomerPhoneNumber(customerPhoneNumber);

return customerInfo;
}
/**
* getEntityContext method comment
* @return javax.ejb.EntityContext
*/
public javax.ejb.EntityContext getEntityContext() {
return entityContext;
}

/**
* Refreshes the enterprise bean corresponding to the primary key from the persistent
* storage.
* Creation date: (11/14/2001 2:06:39 PM)
* @param aPrimaryKey com.flights.ejb.bmp.FlightKey
```

```
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.FinderException The exception description.
*/
private void refresh(CustomerKey primaryKey) throws java.rmi.RemoteException, javax.ejb.FinderException {

DocumentCollection customerDC = null;
Document customerDoc = null;

System.out.println("starting refresh");

if (primaryKey == null)
throw new RemoteException("Primary key cannot be null");

customerNumber = primaryKey.primaryKey;
System.out.println("Customer Number in refresh: " + customerNumber);

try {

ndbContent = getConnection();

// find the customer number to obtain customer info
customerDC = ndbContent.search("SELECT (Form = \"FlightPerson\") & (PersonalID = "+ customerNumber.trim()
+")");
System.out.println("number of elements in collection: " + customerDC.getCount());
customerDoc = customerDC.getFirstDocument();

if (customerDoc != null) {
customerFirstName = customerDoc.getItemValueString("FirstName");
customerMiddleInitial = customerDoc.getItemValueString("MiddleInitial");
customerLastName = customerDoc.getItemValueString("LastName");
customerAddress = customerDoc.getItemValueString("StreetAddress");
customerCity = customerDoc.getItemValueString("City");
customerState = customerDoc.getItemValueString("State");
customerZipCode = customerDoc.getItemValueString("Zip");
customerCountry = customerDoc.getItemValueString("Country");
customerPhoneNumber = customerDoc.getItemValueString("PhoneNumber");
customerInternetAddress = customerDoc.getItemValueString("InternetAddress");

System.out.println("FN: " + customerFirstName + " LN: " + customerLastName + " MI: " + customerMiddleInitial +
" Internet Addr: " + customerInternetAddress);

} // end if
else
{
throw new FinderException("Customer information not found for customer number " + customerNumber);

}

} // end try
catch(lotus.domino.NotesException ne){
System.out.println(ne.text + " " + ne.id);
ne.printStackTrace();
throw new RemoteException(ne.toString());
}
catch(Exception e) {
e.printStackTrace();
throw new RemoteException(e.toString());
}
finally
{
```

```
try
{
System.out.println("Recycle objects in CustomerBean refresh");
if (customerDC != null)
customerDC.recycle();
if (customerDoc != null)
customerDoc.recycle();


}
catch(Exception ex)
{
throw new RemoteException(ex.toString());
}
}
}
/**
* Sets the address for the customer.
* Creation date: (04/02/02 2:37:06 PM)
* @param newCustomerAddress java.lang.String
*/
public void setCustomerAddress(java.lang.String newCustomerAddress) {
customerAddress = newCustomerAddress;
}
/**
* Sets the city for the customer.
* Creation date: (04/02/02 2:37:28 PM)
* @param newCustomerCity java.lang.String
*/
public void setCustomerCity(java.lang.String newCustomerCity) {
customerCity = newCustomerCity;
}
/**
* Sets the country for the customer.
* Creation date: (04/02/02 2:37:46 PM)
* @param newCustomerCountry java.lang.String
*/
public void setCustomerCountry(java.lang.String newCustomerCountry) {
customerCountry = newCustomerCountry;
}
/**
* Sets the first name of the customer.
* Creation date: (04/02/02 2:38:05 PM)
* @param newCustomerFirstName java.lang.String
*/
public void setCustomerFirstName(java.lang.String newCustomerFirstName) {
customerFirstName = newCustomerFirstName;
}
/**
* Sets the internet address for the customer.
* Creation date: (04/02/02 2:38:24 PM)
* @param newCustomerInternetAddress java.lang.String
*/
public void setCustomerInternetAddress(java.lang.String newCustomerInternetAddress) {
customerInternetAddress = newCustomerInternetAddress;
}
/**
* Sets the last name of the customer.
* Creation date: (04/02/02 2:38:39 PM)
* @param newCustomerLastName java.lang.String
```

```
*/
public void setCustomerLastName(java.lang.String newCustomerLastName) {
customerLastName = newCustomerLastName;
}
/**
* Sets the middle initial of the customer.
* Creation date: (04/02/02 2:38:55 PM)
* @param newCustomerMiddleInitial java.lang.String
*/
public void setCustomerMiddleInitial(java.lang.String newCustomerMiddleInitial) {
customerMiddleInitial = newCustomerMiddleInitial;
}
/**
* Sets the customer number.
* Creation date: (04/02/02 2:40:12 PM)
* @param newCustomerNumber java.lang.String
*/
public void setCustomerNumber(java.lang.String newCustomerNumber) {
customerNumber = newCustomerNumber;
}
/**
* Sets the phone number for the customer.
* Creation date: (04/02/02 2:39:10 PM)
* @param newCustomerPhoneNumber java.lang.String
*/
public void setCustomerPhoneNumber(java.lang.String newCustomerPhoneNumber) {
customerPhoneNumber = newCustomerPhoneNumber;
}
/**
* Sets the state for the customer.
* Creation date: (04/02/02 2:39:22 PM)
* @param newCustomerState java.lang.String
*/
public void setCustomerState(java.lang.String newCustomerState) {
customerState = newCustomerState;
}
/**
* Sets the zip code for the customer.
* Creation date: (04/02/02 2:39:40 PM)
* @param newCustomerZipCode java.lang.String
*/
public void setCustomerZipCode(java.lang.String newCustomerZipCode) {
customerZipCode = newCustomerZipCode;
}
/**
* setEntityContext method comment
* @param ctx javax.ejb.EntityContext
* @exception java.rmi.RemoteException The exception description.
*/
public void setEntityContext(javax.ejb.EntityContext ctx) throws java.rmi.RemoteException {
entityContext = ctx;
}
/**
* unsetEntityContext method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void unsetEntityContext() throws java.rmi.RemoteException {
entityContext = null;
}
}
```

The source code for the CustomerInfoBean is shown in Example 5.

**Example 5:** CustomerInfoBean source code

```
package com.flights;
import java.util.*;

/**
* CustomerInfo is a JavaBean used to store the customer information for
* a specific customer/flight. It is passed to the appropriate
* JSP which will use it to retrieve the specific customer data.
*/
public class CustomerInfo implements java.io.Serializable {

private java.lang.String customerFirstName;
private java.lang.String customerMiddleInitial;
private java.lang.String customerLastName;
private java.lang.String customerAddress;
private java.lang.String customerCity;
private java.lang.String customerState;
private java.lang.String customerZipCode;
private java.lang.String customerCountry;
private java.lang.String customerPhoneNumber;
private java.lang.String customerInternetAddress;
private Vector customerListVector;
/**
* CustomerInfo constructor.
*/
public CustomerInfo() {
super();
}
/**
* Returns address for customer.
* Creation date: (02/11/02 10:08:49 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerAddress() {
return customerAddress;
}
/**
* Returns city for customer.
* Creation date: (02/11/02 10:10:19 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerCity() {
return customerCity;
}
/**
* Returns country for customer.
* Creation date: (02/11/02 10:11:38 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerCountry() {
return customerCountry;
}
/**
* Returns first name of customer.
* Creation date: (02/11/02 10:28:15 AM)
* @return java.lang.String
```

```
*/
public java.lang.String getCustomerFirstName() {
return customerFirstName;
}
/**
* Returns internet address for customer.
* Creation date: (04/02/02 12:31:05 PM)
* @return java.lang.String
*/
public String getCustomerInternetAddress() {
return customerInternetAddress;
}
/**
* Returns last name of customer.
* Creation date: (02/11/02 10:29:22 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerLastName() {
return customerLastName;
}
/**
* Returns list of customers.
* Creation date: (02/11/02 10:30:00 AM)
* @return java.lang.String
*/
public Vector getCustomerListVector() {
return customerListVector;
}
/**
* Returns middlet initial of customer.
* Creation date: (02/11/02 10:30:00 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerMiddleInitial() {
return customerMiddleInitial;
}
/**
* Returns phone number for customer.
* Creation date: (02/11/02 10:30:28 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerPhoneNumber() {
return customerPhoneNumber;
}
/**
* Returns state for customer.
* Creation date: (02/11/02 10:31:01 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerState() {
return customerState;
}
/**
* Returns zip code for customer.
* Creation date: (02/11/02 10:31:32 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerZipCode() {
return customerZipCode;
}
```

```
/**
* Sets address for customer.
* Creation date: (02/11/02 10:42:16 AM)
*/
public void setCustomerAddress(java.lang.String newCustomerAddress) {
customerAddress = newCustomerAddress;
}
/**
* Set city for customer.
* Creation date: (02/11/02 10:53:31 AM)
*/
public void setCustomerCity(java.lang.String newCustomerCity) {
customerCity = newCustomerCity;
}
/**
* Sets country for customer.
* Creation date: (02/11/02 10:57:06 AM)
*/
public void setCustomerCountry(java.lang.String newCustomerCountry) {
customerCountry = newCustomerCountry;
}
/**
* Sets first name of customer.
* Creation date: (02/11/02 10:57:31 AM)
*/
public void setCustomerFirstName(java.lang.String newCustomerFirstName) {
customerFirstName = newCustomerFirstName;
}
/**
* Sets internet address for customer.
* Creation date: (04/02/02 12:32:18 PM)
* @param newCustomerInternetAddress java.lang.String
*/
public void setCustomerInternetAddress(String newCustomerInternetAddress) {
customerInternetAddress = newCustomerInternetAddress;
}
/**
* Sets last name of customer.
* Creation date: (02/11/02 10:57:57 AM)
*/
public void setCustomerLastName(java.lang.String newCustomerLastName) {
customerLastName = newCustomerLastName;
}
/**
* Sets list of customers.
* Creation date: (02/11/02 10:58:12 AM)
*/
public void setCustomerListVector(Vector newCustomerListVector) {
customerListVector = newCustomerListVector;
}
/**
* Sets middle initial of customer.
* Creation date: (02/11/02 10:58:39 AM)
*/
public void setCustomerMiddleInitial(java.lang.String newCustomerMiddleInitial) {
customerMiddleInitial = newCustomerMiddleInitial;
}
/**
* Sets phone number for customer.
* Creation date: (02/11/02 10:58:39 AM)
```

```
*/
public void setCustomerPhoneNumber(java.lang.String newCustomerPhoneNumber) {
customerPhoneNumber = newCustomerPhoneNumber;
}
/**
* Sets state for customer.
* Creation date: (02/11/02 10:58:52 AM)
*/
public void setCustomerState(java.lang.String newCustomerState) {
customerState = newCustomerState;
}
/**
* Sets zip code for customer.
* Creation date: (02/11/02 10:59:08 AM)
*/
public void setCustomerZipCode(java.lang.String newCustomerZipCode) {
customerZipCode = newCustomerZipCode;
}
}
```

# WebSphere(R) Application Server and Lotus(R) Domino(TM) interoperability overview

The iSeries(TM) System Test flights scenario consisted of a a WebSphere Application Server front end with Lotus Domino databases on the backend. As a result, we needed to ensure that the WebSphere Application Server front end could work with the Lotus Domino back end. This included the ability to communicate between the two environments, share a single sign-on (SSO) across the environments, and ensure security within each of the environments.

To communicate between the two environments, the flights application used WebSphere Application Server programs written in Java(TM) that used the Lotus Domino Java APIs. The Lotus Domino workflow managed the population of the flights. The WebSphere Application Server transaction services were used to implement the customer flight Web site.

The flights application uses the Lotus Domino LDAP (Lightweight Directory Access Protocol) directory. The LDAP directory is needed for implementing WebSphere Application Server security. LDAP allows the flight customers to authenticate using customer numbers and passwords. The flights application validates the user's identity to allow access to customer and payment information.

The flights application uses the single sign-on capability that spans WebSphere Application Server and Lotus Domino. This allows the flight application to access either while only requiring the user to sign on once. This means that when a transaction is initiated from WebSphere Application Server to the Lotus Domino database, the same login credentials are used. The opposite is also possible, but the flights application did not have a need for Lotus Domino to access WebSphere Application Server. The goal of single sign-on is to provide a seamless flow of information across the products.

# WebSphere(R) Application Server and Lotus(R) Domino(TM) interoperability single sign-on

For the flights application, security is a concern. Both WebSphere Application server and Lotus Domino provide support for securing access and data. Both products implement security mechanisms which involve determining and verifying user identity (authentication) and allowing access to protected resources to designated users (authorization).

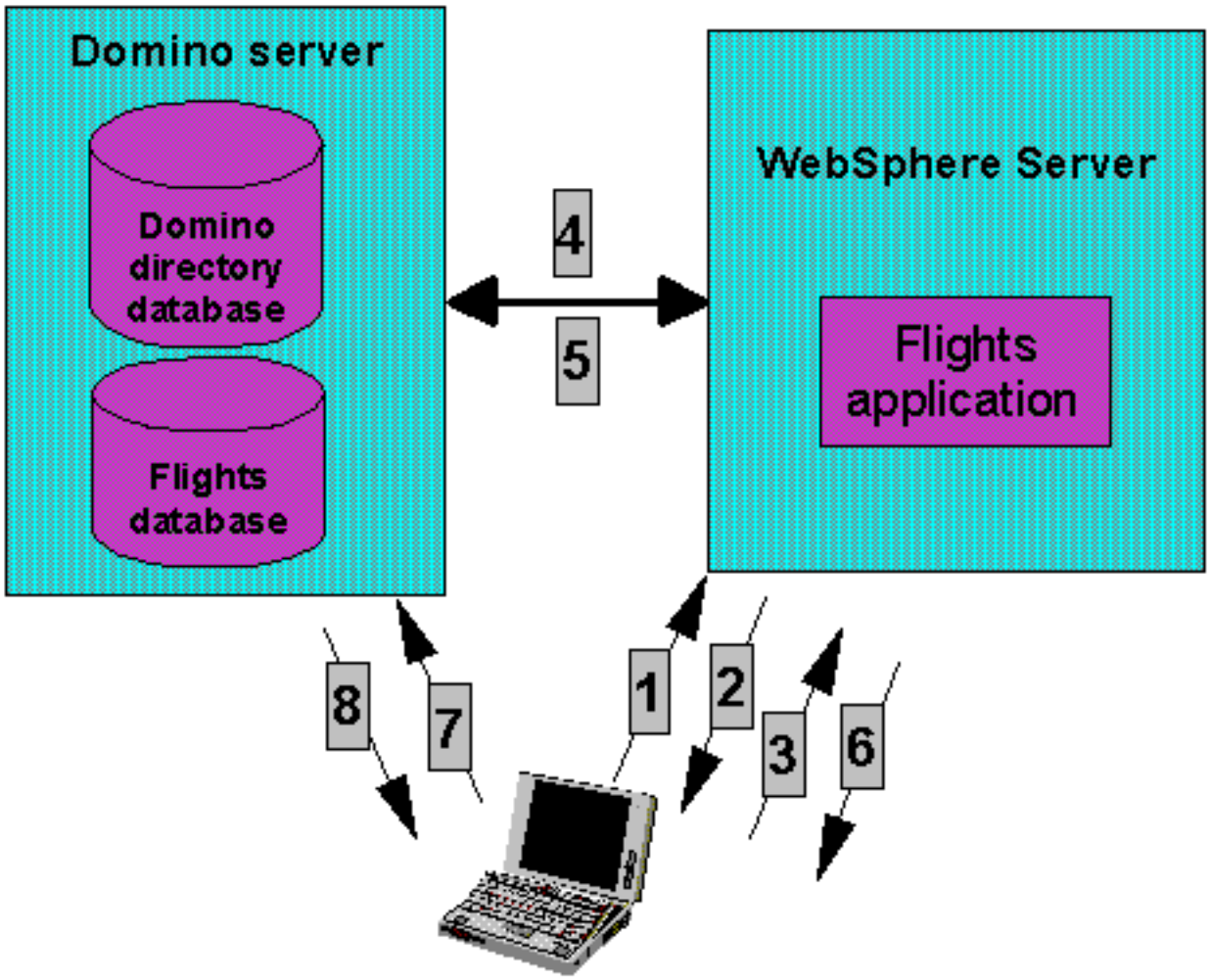
Using single sign-on (SSO), the flight's Web users can authenticate once to the WebSphere Application Server, and then access Lotus Domino without logging in again. This is accomplished by configuring the Lotus Domino and WebSphere Application Servers to share authentication information gathered from the single Lotus Domino LDAP server.

To enable SSO between servers, the Lightweight Third Party Authentication (LTPA) mechanism is used. This mechanism utilizes an LTPAToken which contains the user authentication information, the network domain in which the SSO is valid, and the expiration time. The LTPAToken is encrypted using the LTPA keys that must be shared for all the SSO participating servers.

The token is issued to the Web user in a cookie. This cookie resides in browser memory and is not stored on the user's computer and expires when the user closes the browser.

Enabling single sign-on (SSO) on WebSphere, requires configuring the Enterprise Application Resource (EAR) file for security and configuring the global security settings in WebSphere Application Server. Figure 1 shows the flight application single sign-on process between Lotus Domino and WebSphere Application Server.

**Figure 1 Lotus Domino and WebSphere SSO**




1. A Web user submits a request to the Web server (HTTP server) for a protected resource, to obtain the home page.
2. The Web server prompts the user for the authentication information.
3. The user responds by supplying the information (customer number and password).
4. Then the Web server contacts the LTPA server (WebSphere Application Server) which connects with the Lotus Domino Directory to verify the authentication information.
5. If the information supplied for the user is correct, Lotus Domino responds to the server (WebSphere Application Server) with the validated information.
6. The server uses the returned values to check if the user has access to the requested resource and issues an LTPA token for the user. The Web server sends the token to the user as a HTTP Cookie which is stored in the user's browser and serves the requested resource (index.html).
7. Once the user is authenticated and the cookie is available, they can request another protected resource from Lotus Domino or WebSphere Application Server.
8. Lotus Domino and WebSphere Application Server validate the token provided for the user and tell the Web server to send the requested resource to the browser, as long as the user has enough access to that resource, without prompting again with the challenge information.

## Setting up IIOP on Lotus Domino

Remote Lotus Domino objects make use of Common Object Request Broker Architecture (CORBA) to implement access to Lotus Domino. In CORBA, communication between objects occurs through Object Request Brokers (ORBs) that use the Internet Inter-ORB Protocol (IIOP) to send messages to each other. In Lotus Domino, the Lotus Domino Internet Inter-ORB Protocol (DIIOP) service is used for CORBA communication. The advantage of using remote

objects is that the Web application server can run on a separate machine from the Lotus Domino server. The code imports the lotus.domino.* package. The NotesFactory() method requires a Lotus Domino server IP address or name and a valid user ID and profile that has access to IIOP and the Lotus Domino server. The remote Lotus Domino server needs to have DIIOP configured and the user ID must be authorized to use the Lotus Domino server and be authorized to run IIOP agents.

The flights application used remote Lotus Domino objects when using the Lotus Domino Java(TM) APIs for the following reasons:

- Configuration and execution time requirements are simpler
- No restrictions on usage in multi-threaded environments

To configure the Lotus Domino server for CORBA, perform the following steps:

1. Edit the notes.ini file and add the tasks to the lists of tasks specified in the ServerTasks parameter. For example:
   ServerTasks=<any other tasks>, HTTP, DIIOP
2. Open the Lotus Domino server document for editing and choose the Ports -> Internet Ports -> IIOP tab. Set the port number and enable the port. The default port number is 63149 for a TCP/IP IIOP port and 63148 for the IIOP port that uses SSL. Also, you can specify whether to allow name and password and anonymous access.
3. Configure the number of threads and time-out value. The time-out value represents the number of minutes a connection can be idle before being dropped by the server. To set the time-out value, choose Internet Protocols -> IIOP tab of the Lotus Domino server document.
4. Set security for accessing the Lotus Domino server and for running Java programs on the server. In the Server access section, specify who can access the server. If the field is left blank, no one is denied access to the server. If the field has any names listed in it, then only those people or groups specifically listed can access the server. If users are required to log in, this information is checked by the server after they have been authenticated.
5. In the Java/COM Restrictions section of the Lotus Domino server document, specify the users who are allowed to access the Lotus Domino objects using CORBA. If this field is left blank, no one is allowed to access the Lotus Domino objects using CORBA. The field can accept wildcards. An asterisk (*) in this field allows everyone.
6. If your Lotus Domino server is behind a firewall, edit the Lotus Domino server's Notes(TM).ini and add the line DIIOP_IOR_HOST=ipAddress. Here ipAddress is the IP address of your Lotus Domino server as it is known outside of the firewall.

## Configuring the flights application for security

Before configuring WebSphere Application Server or Lotus Domino, the application needs to be configured for security and installed. For information on configuring an application for security, see the IBM(R) Redbook, IBM WebSphere V4.0 Advanced Edition Security. The chapters on Securing Web Components and Securing Enterprise Bean Components were followed for securing the flights application in WebSphere Studio Application Developer.

In the flights application, Form-based authentication was used. This authentication mechanism allowed the flights site to specify a site specific HTML login page. When using Form-based authentication, the password is not encrypted and the target server is not authenticated which provides a security risk. To avoid this security risk, secure transport (SSL) could be used.

## Enabling single sign-on for WebSphere Application Server

Configuring Global Security Settings for SSO in WebSphere involves the following:

1. Start WebSphere Administrator's Console.
2. Select Console -> Security Center. This will display the global security settings for WebSphere. Check Enable Security in the General tab.

3. Click on the Authentication tab and choose Lightweight Third Party Authentication (LTPA) as the Authentication mechanism type.
4. Specify the following LTPA settings:
    1. How many minutes can pass before a client using an LTPA token must authenticate again in the Token Expiration field.
    2. Check Enabled single sign-on (SSO). The Domain field will then be enabled.
    3. Enter a DNS domain name in the Domain field. This domain name is used when the HTTP cookie is created for SSO and determines the scope to which SSO applies.

**Important:** All SSO participating servers must be in the same DNS domain.

5. Check the LDAP radio button and input the LDAP server settings.

**Note:** Make sure to configure WebSphere Application Server to use Lotus Domino 5.0 as the Directory type in Security Center. Also make sure that the Lotus Domino server is running and the LDAP task is started, because the Security server ID and password will be verified.

6. Click on the Generate Keys button to create the LTPA keys for encrypting the LTPA token. You will be prompted for an LTPA password to protect the set of encryption keys. These LTPA keys must be shared for all servers using SSO.

**Important:** Remember that the generation of the LTPA keys must be done when the Lotus Domino LDAP server settings are configured. This guarantees that the LDAP host name and port are present in the exported file. Lotus Domino needs this information during the Web SSO configuration document creation process.

7. Once the LTPA keys are generated, click on the Export Key button to export the LTPA keys to a file. This file is used to import the keys into Lotus Domino.
8. Click on Apply and then OK.
9. When the process is completed a warning message will display, saying: Changes will not take effect until the admin server is restarted. Click OK.
10. Restart the administration server by selecting the node included in the node folder located in the tree view on the left side of the console and then right-clicking on it and select Restart in the resulting context menu.

## Enabling single sign-on for the Lotus Domino Server

1. Create a new Web SSO configuration Document in the Lotus Domino Directory database.
    1. Select Server -> Servers to display the view. Click on the Web button and select Create Web SSO Document in the resulting context menu.
    2. A new document will be displayed with the following Token Name field (LTPAToken).
    3. Include the DNS domain in the Token Domain field. This value must coincide with the value specified in the Domain field in WebSphere Application Server. This domain name is used when the HTTP cookie is created for single sign-on and determines the scope to which single sign-on applies.
    4. Choose the Lotus Domino servers that are going to participate in the SSO scenario.

    **Note:** You must specify a fully qualified Lotus Domino server name (for example, MyDominoServer/MyOu). The Lotus Domino server name that you specify must also match the name of the Home/mail server currently in the active Location document on your Lotus Notes(R) client. If the Location document does not match, you must create one that does.

    5. Enter the maximum number of minutes that the issued token will be valid in the Expiration (minutes) files. Set it to match the time set in WebSphere Application Server.
    6. Click on the Keys drop down and select Import WebSphere LTPA keys.
    7. Specify the path and the file name for the WebSphere Application Server LTPA keys file exported earlier.

8. Click OK. A new dialog box will appear prompting the user for the LTPA password specified when the keys were generated.
9. Click OK. When the process completes a confirmation message will be displayed.
10. A new WebSphere Information section will appear in the document. The LDAP realm and host name are read from the WebSphere Application Server Import file. If a port was specified in the WebSphere LDAP configuration setting, make sure to add a blackslash (\) prior to the colon (:) in the LDAP Realm field.
11. Click on Save and Close button. The document will be saved. To check if the document is present in the Lotus Domino Directory select Server -> Web Configurations View and expand the *-All Servers - section. The new document should be displayed as Web SSO configuration for LtpaToken.

2. Enable TCP/IP port status in Ports -> Internet Ports -> Web tab and do not allow anonymous connections over TCP/IP by modifying the Lotus Domino Server Document.
3. Select Multi-Server session in Internet Protocols -> Lotus Domino Web Engine tab on the server document.
4. Select More name variations with lower security in the Web server authentication section of the security tab. This allows users to enter the following name formats in the name and password dialog box: last name, first name, common name, full hierarchical name, short name, and alias name.

Table 1 shows how the Database ACL was modified to implement security on the flights application database.

**Table 1 Lotus Domino and WebSphere SSO**

| People, Servers, and Group | User Type | Access Level | Authorization |
|---|---|---|---|
| Flight Employees | Person Group | Editor | Delete Documents, Create Lotus Script/Java Agent |
| Anonymous | Unspecified | Reader | Write Public Documents |

The Lotus Domino server will present a default server login page when a user tries to access a Web page from the database. The Web user then needs to enter their username and password and click on the Login button.

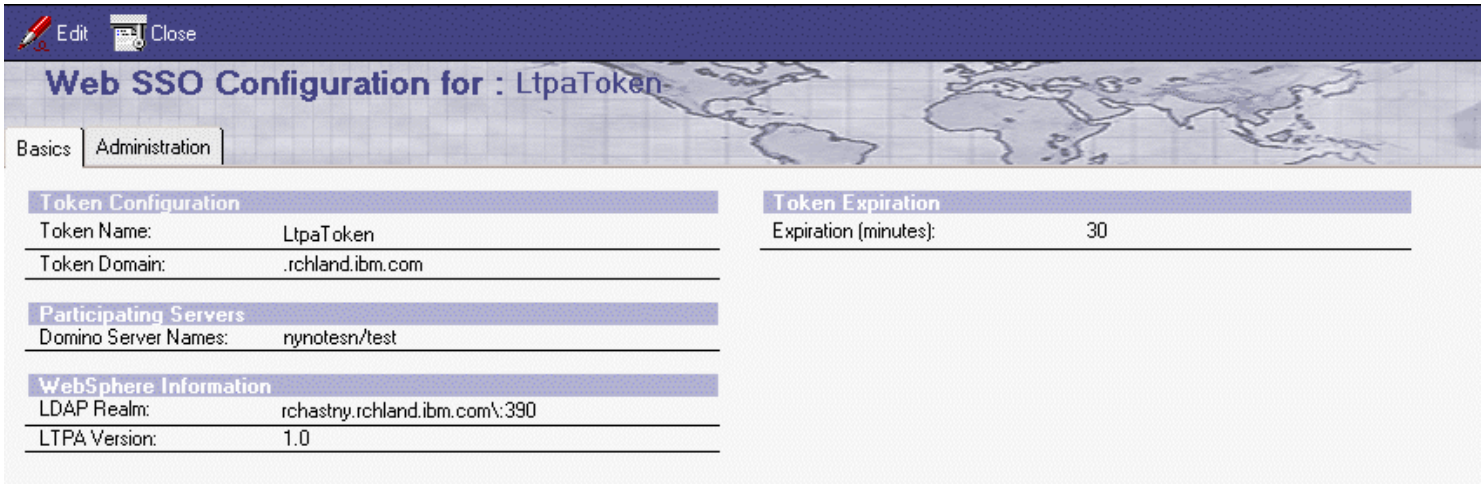
The Lotus Domino server checks if the user is registered in the Lotus Domino Directory database and verifies that the credential values are correct. It also checks if the user has access to the database. Once the user is authenticated, Lotus Domino creates a new LTPAToken and sends it to the user as a HTTP cookie and opens the Lotus Domino document.

# WebSphere(R) Application Server and Lotus(R) Domino(TM) inter operability key findings

Following is a list of key findings that we uncovered while implementing the flights scenario which dealt with WebSphere Application Server and Lotus Domino inter operability.

- When creating the Lotus Domino Web single sign-on (SSO) configuration document, you need to make sure the location document of your Notes(TM) client points to the Lotus Domino server where you want to enable SSO. This is needed so that a public key can be used for the server. If a message appears when you save the Web SSO Configuration document saying it could not find server, then this should fix the message.
- In the WebSphere Application Server security center, the realm (domain name) needs to be specified in lower case.
- Since WebSphere Application Server treats the DNS name as case-sensitive, ensure that the DNS domain value is specified exactly the same, including casing as in Figure 1, whenever you use this value in Lotus Domino.

**Figure 1 Lotus Domino and WebSphere SSO**




- Make sure that you fully qualify the URL with the domain name (ie http://*systemName.domainName:portNumber/uri*) when accessing either a Lotus Domino or a WebSphere URL when security and SSO are configured. If you do not fully qualify the URL (ie http://*systemName:portNumber/uri*), you will get returned to the login form and never get to the page you were trying to access.
- When security is configured in WebSphere Application Server, place the NCSOW.jar in the qibm/userdata/webasadv4/*instanceName*/lib/ext directory. Because we were using the NCSOW.jar file to create our Lotus Domino sessions, we had put this jar file in the JVM properties of the default server. This caused a problem once security was enabled. With the jar file in the JVM settings, we kept getting a NoClassDefFound for com.ibm.ejs.oa.EJSORB. Since the NCSOW was in the JVM classpath we were seeing this problem. To fix, we moved the NCSOW.jar to the qibm/userdata/webasadv4/*instanceName*/lib/ext directory and removed the entry from the JVM classpath.
- When using security, you can verify that you are running under the correct identity by using the getCalledIdentity() method on the enterprise bean Context. This method will print out the user identity that the method inside the enterprise bean is running under.
- The following Java/CORBA class (lotus.domino package) elements support sign-on to Lotus Domino and WebSphere servers in a single sign-on domain.
  - SessionToken property
    Read-only. Gets a session token for enabling sign-on to Lotus Domino and WebSphere servers in a domain that supports single sign-on.
    *NOTE:* This property is new with R5.0.5.
    *Defined in:* lotus.domino.Session
    *Data type:* String
    *Syntax:* public String getSessionToken() throws NotesException
    *Usage:* The token is unique for each user and is valid for the time specified in the Lotus Domino Directory. The format of the token is consistent with the LtpaToken cookie used by WebSphere.

    You can also get the token from the HTTP headers in a servlet with HttpServletRequest.getCookies().
    This property is valid only on a server configured for single sign-on.
    See NotesFactory for usage and examples.
  - NotesFactory class
    *NOTE:* To make remote (IIOP) calls to the Lotus Domino Objects in a WebSphere environment, NCSOW.jar must be in your classpath. This is new with R5.0.4.
    The description of the NotesFactory class is extended as follows.
    *NOTE:* These extensions are new with R5.0.5.

    To access a server using single sign-on, create a Session object as follows. For remote (IIOP) calls, the first parameter is the Internet name of the host. For local calls, the first parameter is null.
    - createSession(hostString, String token) - Access is granted based on the token. This method works in a Lotus Domino environment. The token must be a valid token for single sign-on obtained from Session.getSessionToken or the LtpaToken cookie used by WebSphere.
    - createSession(hostString, org.omg.SecurityLevel2.Credentials) - Access is based on the Credentials object. This method works in a WebSphere environment where the Credentials object is created using loginHelper.
    - createSession(hostString, null) - Access is granted based on the current Credentials object in the WebSphere environment. This method works from an enterprise bean application in WebSphere.

    The specification of NotesFactory is extended with the following methods:
    - static public Session createSession(String host, String token) throws NotesException
    - static public Session createSession(String host, org.omg.SecurityLevel2.Credentials) throws NotesException
  - Examples
    - **Example 1:** This Lotus Domino agent gets a token for single sign-on and creates a remote (IIOP) session to another server based on the token.

```
import lotus.domino.*;
public class JavaAgent extends AgentBase {

public void NotesMain() {
try {
Session session = getSession();
AgentContext agentContext = session.getAgentContext();
Session s2 = NotesFactory.createSession("test5.iris.com",
session.getSessionToken());
System.out.println("remote session name = " + s2.getUserName());
} catch(Exception e) {
e.printStackTrace();
}
}
}
```

- **Example 2:** This servlet gets a token for single sign-on from the LTPAToken cookie through HttpServletRequest and creates a session based on the token.

```
import java.lang.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;

public class Cookies extends HttpServlet
{

private void respond(HttpServletResponse response, String entity) throws IOException
{
response.setContentType("text/plain");
if (entity == null)
{ response.setContentLength(0);}
else {
response.setContentLength(entity.length() + 1);
ServletOutputStream out = response.getOutputStream();
out.println(entity);
}
}

public void doGet (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
String s1 = "";
Cookie[] cookies = null;
String sessionToken = null;

try {
cookies = request.getCookies();
}
catch (Exception e) {
respond(response,"Exception from request.getCookies(): " + e.toString());
return;
}

if (cookies == null) {
s1 = "No cookies received";
}
else {
for (int i = 0; i < cookies.length; i++) {
if (cookies[i].getName().equals("LtpaToken")) {
sessionToken = cookies[i].getValue();
}
}
}

if (sessionToken != null) {
try {
NotesThread.sinitThread();
Session session = NotesFactory.createSession(null, sessionToken);
s1 += "\n" + "Server: " + session.getServerName();
s1 += "\n" + "IsOnServer: " + session.isOnServer();
s1 += "\n" + "CommonUserName: " + session.getCommonUserName();
s1 += "\n" + "UserName: " + session.getUserName();
s1 += "\n" + "NotesVersion: " + session.getNotesVersion();
s1 += "\n" + "Platform: " + session.getPlatform();
NotesThread.stermThread();
}
catch (NotesException e) {
s1 += "\n" + e.id + e.text;
e.printStackTrace();
}
}

respond(response,s1);
}
}
```

- **Example 3:** This application snippet creates a session based on a credentials object obtained from WebSphere.

```
com.ibm.CORBA.iiop.ORB orb = com.ibm.ejs.oa.EJSORB.getORBInstance();

if (orb != null) {
org.omg.SecurityLevel2.Current(R) securityCurrent = (org.omg.SecurityLevel2.Current)orb.resolve_initial_references("SecurityCurrent");
org.omg.SecurityLevel2.Credentials invCred = securityCurrent.get_credentials(org.omg.Security.CredentialType.SecInvocationCredentials);
System.out.println("creating notes session using current creds");
session = NotesFactory.createSession(notesServer, invCred);
}
```

- **Example 4:** This WebSphere enterprise bean application creates a session based on the current credentials object in the WebSphere environment.

```
import lotus.domino.*;

public class HelloBean extends Object implements SessionBean {

... /* See HelloBean.java from Websphere for the complete class code */

/**
Returns the greeting. But has been modified to create a remote session to the
Lotus Domino server.
@return The greeting.
@exception RemoteException Thrown if the remote method call fails.
```

```
*/
public String getMessage () throws RemoteException
{
String result = "hello bean ";

try {
Session s = NotesFactory.createSession("test5.iris.com", null);
result = result + " -- Got Session for " + s.getUserName();
}
catch (NotesException ne) {
result = result + "-- " + ne.text;
result = result + "-- failed to get session for user";
}

return (String) result + " -- done";
}
}
```

# References

The following resources provide information you may find helpful.

- IBM(R) WebSphere V4.0 Advanced Edition Security, IBM Redbook SG24-6520-00
- Lotus Domino and WebSphere Integration on the IBM eServer iSeries(TM) Server, IBM Redbook SG24-6223
- Lotus Domino and WebSphere Together Second Edition, IBM Redbook SG24-5955-01
- Security Guide
  http://www.ibm.com/software/webservers/appserv/doc/v40/aes/infocenter/was/pdf/nav_Securityguide.pdf
- WebSphere Inter operability between Versions 3.5.x and 4.0.x
  http://www7b.software.ibm.com/wsdd/library/techarticles/0202_sundman/sundman.html