Building E-commerce Solutions: Business to Consumer

Delivery Guide

Course Number: 1588A
## Contents

### About This Course
- Course Overview ................................................................. v
- Course Timing ........................................................................ vi
- Document Conventions .......................................................... vii

### Introduction
- Introductions ............................................................................. 3
- Course Materials ........................................................................ 4
- Prerequisites .............................................................................. 5
- Course Outline: Day One .......................................................... 6
- Course Outline: Day Two .......................................................... 7
- Microsoft Certified Professional Program ...................................... 8
  - MCSD Track ........................................................................... 10
- Facilities .................................................................................... 13

### Module 1: Understanding Electronic Commerce
- Overview ................................................................................ 17
- Overview of E-Commerce ......................................................... 18
  - What Is E-Commerce? .............................................................. 19
  - Why Use E-Commerce? ........................................................... 20
  - How Does E-Commerce Work? .............................................. 21
- Tour of a Sample Site ............................................................... 22
  - Requirements of a Good E-Commerce Site ......................... 23
- Demonstration: Visiting the Sample Site ....................................... 25
  - A Few E-Commerce Sites on the Net ...................................... 28
- Introduction to Commerce Server .............................................. 29
  - Overview of Commerce Server ............................................. 30
  - Commerce Server Platform Architecture ................................ 32
  - Important Commerce Server Tools ........................................ 33
- Review ..................................................................................... 35

### Module 2: Building a Site
- Overview ................................................................................ 39
- Creating a Site Foundation ....................................................... 40
  - What the Site Foundation Wizard Creates ............................... 41
  - Prepare a Database for the Site .............................................. 42
- Demonstration: Using the Site Foundation Wizard ......................... 43
- Creating a Site .......................................................................... 45
  - What the Site Builder Wizard Generates ................................ 46
- Demonstration: Creating and Testing a Site ................................. 47
- Working with the Server Administration Pages ............................. 50
  - Opening, Closing, and Deleting Sites .................................... 51
  - Modifying the Properties of a Site ....................................... 52
  - Managing a Site ................................................................. 53
- Lab 2: Creating a New Commerce Server Site ............................... 54
- Review ..................................................................................... 55

### Module 3: Enhancing the Product Catalog
- Overview ................................................................................ 59
- Introducing Commerce Server Objects ....................................... 60
Stages in the Plan Pipeline ................................................................. 135
Components in the Plan Pipeline Stages ...................................... 137
Review ......................................................................................... 139

Module 6: Checking Out
Overview .................................................................................. 141
Capturing Shopper Information .................................................. 142
  Understanding Checkout ......................................................... 143
Practice: Customizing the Shipping Page ................................ 144
Computing Order Value .............................................................. 147
  Computing Shipping and Handling Charges ......................... 148
  Computing Tax ...................................................................... 150
  Computing Order Total ......................................................... 152
Adding a Scriptor Component ..................................................... 153
  Understanding the Scriptor Component ................................. 154
  Adding a Scriptor Component to Compute Tax ...................... 156
Lab 6: Adding a Scriptor Component ......................................... 158
Review ......................................................................................... 159

Module 7: Completing the Purchase Process
Overview .................................................................................. 165
Understanding Purchase ............................................................ 166
  Overview of the Purchase Process ........................................... 167
  Capture Payment Information ................................................. 168
Executing the Purchase OPP ....................................................... 169
  Overview of the Purchase OPP ................................................. 170
  Stages of the Purchase OPP ................................................... 171
Tracking an Order ..................................................................... 173
  How to Generate a Simple Order Number? ........................... 174
  How to Track an Order? .......................................................... 175
  Practice: Generating a Simple Order Number ....................... 176
Securing Business Transactions ................................................ 179
  Introducing HTTPS ................................................................. 180
  Introducing Digital Certificates .............................................. 181
Lab 7: Tracking Order Status .................................................... 182
Review ......................................................................................... 183

Module 8: Tracking Shopper Information
Overview .................................................................................. 187
Using Cookies to Track Shoppers .............................................. 188
  Why Track Shoppers? ............................................................. 189
  Setting a Cookie .................................................................. 190
  Retrieving Shopper Information from a Cookie .................... 191
Practice: Using Cookies to Track Shoppers ............................... 192
Using Registration Table to Track Shoppers ......................... 195
  Creating a Shopper Table ...................................................... 196
  Retrieving Shopper Information .......................................... 198
  Modifying the Purchase Pipeline ........................................... 200
Lab 8: Using Registration Table to Track Shoppers ................. 203
Review ......................................................................................... 204
Module 9: Introducing Business-to-Business Commerce

Overview ................................................................. 207
Business-to-Business Commerce ........................................ 208
  Overview of Business-to-Business Commerce ......................... 209
  A Business-to-Business Scenario ........................................ 210
Business Partner Functionality ............................................. 211
  Storing ePartner Information ........................................... 212
  Verifying an ePartner .................................................. 213
  Associating ePartner ID with an Order .................................. 215
  Displaying Orders Received through ePartners ......................... 216
Lab 9: Implementing Business Partner Functionality (Optional) .......... 217
Review ........................................................................ 218
About This Course

Course Overview

This section provides you with a brief description of the course, audience, suggested prerequisites, and course objectives.

Description

This course teaches developers how to implement business-to-customer and business-to-business commerce solutions using Microsoft® Site Server 3.0, Commerce Edition, and the pipeline technology.

Audience

This course is designed for software developers and other IT professionals who design, build, and support commerce-enabled web solutions.

Student Prerequisites

This course requires that students meet the following prerequisites:

- Ability to write client- and server-side script in either VBScript or JScript
- Knowledge of object-oriented programming techniques
- Familiarity with basic Microsoft® SQL Server™ 7.0 commands, such as creating a device, creating a database, and querying a database
- Ability to work with Active Server Pages (ASP) and ActiveX Data Objects (ADO) concepts
- HTML programming skills
- Course 1017, Mastering Web Application Development Using Microsoft® Visual InterDev 6, or equivalent knowledge

Course Objectives

After completing this course, the student will be able to:

- Create the foundation for a commerce-enabled Web site
- Create a simple commerce-enabled Web Site
- Add product data to the site
- Create a business-to-customer order processing pipeline
- Create a custom component for the business-to-customer order processing pipeline
- Create an E-partner for a simple business-to-business order processing pipeline
Course Timing

The following schedule is an estimate of the course timing. Your timing may vary.

Day 1

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00</td>
<td>9:30</td>
<td>Introduction</td>
</tr>
<tr>
<td>9:30</td>
<td>10:25</td>
<td>Module 1: Understanding Electronic Commerce</td>
</tr>
<tr>
<td>10:25</td>
<td>10:35</td>
<td>Break</td>
</tr>
<tr>
<td>10:35</td>
<td>11:30</td>
<td>Module 2: Building a Site</td>
</tr>
<tr>
<td>11:30</td>
<td>11:50</td>
<td>Lab 2: Creating a Site</td>
</tr>
<tr>
<td>11:50</td>
<td>12:30</td>
<td>Lunch</td>
</tr>
<tr>
<td>12:30</td>
<td>2:00</td>
<td>Module 3: Enhancing the Product Catalog</td>
</tr>
<tr>
<td>2:00</td>
<td>2:30</td>
<td>Lab 3: Customizing Product Catalog Page</td>
</tr>
<tr>
<td>2:30</td>
<td>2:40</td>
<td>Break</td>
</tr>
<tr>
<td>2:40</td>
<td>4:00</td>
<td>Module 4: Managing a Shopping Cart</td>
</tr>
<tr>
<td>4:00</td>
<td>5:00</td>
<td>Lab 4: Implementing Upsell</td>
</tr>
</tbody>
</table>

Day 2

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00</td>
<td>9:30</td>
<td>Day 1 review</td>
</tr>
<tr>
<td>9:30</td>
<td>10:15</td>
<td>Module 5: Processing Orders</td>
</tr>
<tr>
<td>10:15</td>
<td>10:25</td>
<td>Break</td>
</tr>
<tr>
<td>10:25</td>
<td>11:25</td>
<td>Module 6: Checking Out</td>
</tr>
<tr>
<td>11:25</td>
<td>11:55</td>
<td>Lab 6: Adding a Scriptor Component</td>
</tr>
<tr>
<td>11:55</td>
<td>12:25</td>
<td>Lunch</td>
</tr>
<tr>
<td>12:25</td>
<td>1:40</td>
<td>Module 7: Completing the Purchase Process</td>
</tr>
<tr>
<td>1:40</td>
<td>2:00</td>
<td>Lab 7: Tracking Order Status</td>
</tr>
<tr>
<td>2:00</td>
<td>3:15</td>
<td>Module 8: Tracking Shopper Information</td>
</tr>
<tr>
<td>3:15</td>
<td>3:35</td>
<td>Lab 8: Tracking Shoppers Using a Registration Database</td>
</tr>
<tr>
<td>3:50</td>
<td>4:30</td>
<td>Lab 9: Implementing E-partner Functionality</td>
</tr>
</tbody>
</table>
### Document Conventions

The following conventions are used in course materials to distinguish elements of the text.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uu</strong></td>
<td>Indicates an overview or introductory page. This symbol appears next to a slide title when additional information on the topic is covered on the page or pages that follow it.</td>
</tr>
<tr>
<td><strong>bold</strong></td>
<td>Represents commands, command options, and portions of syntax that must be typed exactly as shown. It also indicates commands on menus and buttons, dialog box titles and options, and menu names.</td>
</tr>
<tr>
<td><strong>italic</strong></td>
<td>In syntax statements, indicates placeholders for variable information. Italic is also used for book titles.</td>
</tr>
<tr>
<td>Title Capitals</td>
<td>Indicate domain names, user names, computer names, directory names, folders, and file names, except when specifically referring to case-sensitive names. Unless otherwise indicated, you can use lowercase letters when you type a directory name or file name in a dialog box or at a command prompt.</td>
</tr>
</tbody>
</table>

**ALL CAPITALS**

Indicate the names of keys, key sequences, and key combinations—for example, ALT+SPACEBAR.

**`monospace`**

Represents code samples, examples of screen text, or entries that you type at a command prompt or in initialization files.

```
[ ]
```

In syntax statements, enclose optional items. For example, `[filename]` in command syntax indicates that you can choose to type a file name with the command. Type only the information within the brackets, not the brackets themselves.

```
{ }
```

In syntax statements, enclose required items. Type only the information within the braces, not the braces themselves.

```
|   |
```

In syntax statements, separates an either/or choice.

```
➤
```

Indicates a procedure with sequential steps.

```
...
```

In syntax statements, specifies that the preceding item may be repeated.

```
.
```

Represents an omitted portion of a code sample.

```
~
```

Line continuation character. Indicates that ensuing code should be entered without a carriage return.
Instructor Notes: Introduction

Introduction
Presentation: 30 Minutes

This module provides students with an overview of the course content, materials, and logistics for Course 1588A, Building E-commerce Solutions.

Course Materials and Preparation

Materials
To teach this course, you need the following materials:
- Name card
- Student workbook
- Lab manual
- Course Materials compact disc
- Course evaluation
- Reference materials

Preparation
To prepare for this course, you need to:
- Read all course materials.
- Study the review questions at the end of each module and prepare alternative answers.
- Anticipate questions students might ask. Write out these questions and provide answers.
- Successfully set up the classroom.
- Complete all labs and anticipate problems that students may have during the labs. Plan troubleshooting tactics.
- Complete all demonstrations. Practice any whiteboard drawings you anticipate illustrating.
- Practice presenting each module. Spend approximately two minutes per page. Adjust your delivery time accordingly.
- Connect to the Microsoft Online Evaluation System 97 and get familiar with how to use the evaluation system to gather feedback from students. Start by reading the information available online at the following address:
  http://www.microsoft.com/onlineeval/instructions.doc
Check the Microsoft Certified Trainer (MCT) Web site for any posted error logs.

Check the MCT newsgroup for any information about the course, including tips from other MCTs.

Become familiar with the content in the following Microsoft Commerce Server-related Web site:

Module Strategy

Use the following strategy to present this module:

- Course 1588A: Building E-commerce Solutions
  Show the slide that displays the course number and course title before students arrive for class.

- Introductions
  Welcome students to the course and introduce yourself. Provide a brief overview of your background to establish credibility.
  Have students introduce themselves and provide their background, product experience, and expectations of the course.
  Record student expectations on a whiteboard or flip chart that you can reference later in class.

- Course Materials
  Explain the purpose of all materials used in this course.
  Tell students that they will have an opportunity at the end of class to provide feedback on the course and facilities by using the Microsoft Online Evaluation System 97.
  Identify which materials students can keep and which materials are for classroom use only.
  Distribute disks or other materials used in class (if any).

- Prerequisites
  Provide the students with the list of prerequisites that they should have met before taking this course. This is an opportunity for the instructor to identify students who may not have the appropriate background or experience to attend this course.

- Course Outline
  Provide an overview of each module and what students will learn.
  Explain how this course will meet students’ expectations by relating the information covered in individual modules to their expectations.

- Appendixes
  Describe the information provided in any appendixes that may be included with the course materials.

- Microsoft Certified Professional Program
  Inform students about the Microsoft Certified Professional (MCP) program and the various certification options. Also tell students that they will find more certification information on the course compact disc.
- **MCSD Track**
  Explain the recommended certification path for the Microsoft Certified Systems Developer (MCSD) Track.

- **Facilities**
  Inform students of class logistics and rules for the training site.
Introduction
Introductions

**Slide Objective**
To introduce yourself, establish credibility, meet students, and set expectations.

**Lead-in**
Good morning. Welcome to Building E-commerce Solutions.

My name is...

- Name
- Company Affiliation
- Title/Function
- Job Responsibility
- Network Operating System Experience
- Expectations

Introduce yourself.

Provide a brief overview of your background to establish credibility as an e-commerce instructor.

Ask students to introduce themselves, addressing the bulleted items on the slide.

**Delivery Tip**
As students introduce themselves, use a white board or flip chart to record their expectations of the course.
Course Materials

Slide Objective
To identify and describe the course materials and to specify which materials students can keep.

Lead-in
We have provided everything you need for this course. You will find the following materials at your desk...

Name Card
Student Workbook
Lab Manual
Compact Disc
Course Evaluation
Reference Materials

The following materials are included with your kit:

- **Name card.** Write your name on both sides of the name card so that students in front of you and in back of you will know who you are.
- **Student workbook.** The student workbook contains the slide graphics and text covered during lectures. This workbook is yours to keep.
- **Lab manual.** The lab manual contains the hands-on lab exercises and lab answers used during class. This manual is yours to keep.
- **Compact disc.** The compact disc contains a copy of the completed class site as well as starting and ending code for each of the labs.
- **Course evaluation.** Before class is over, please complete the course evaluation to provide feedback on the instructor, course, and software product. Your comments will help us improve future courses.

To provide additional comments on course materials, send e-mail to mstrain@microsoft.com. Be sure to type **Course 1588** in the subject line.

- **Reference materials.** Reference materials, such as product documentation, are for classroom use only.
Prerequisites

To successfully complete this course, you must have the following background:

- Working knowledge of scripting
- Working knowledge of object-oriented programming.
- Working knowledge of SQL, particularly how to create a DSN as well as creating a variety of queries.
- Working knowledge of Active Server Pages (ASP), Microsoft® ActiveX® Data Objects (ADO), and Hypertext Markup Language (HTML) including forms.
- Completion of Course 1017, Mastering Web Application Development Using Microsoft® Visual Interdev, 6.
Module 1, “Understanding Electronic Commerce,” gives an introduction to electronic commerce, Commerce Server, an overview of the shopping process, and a tour of a sample site.

Module 2, “Building a Site,” shows students how to use the Site Foundation Wizard and the Site Builder Wizard to create a new site.

Module 3, “Enhancing the Product Catalog,” shows students how to modify a wizard-generated site and use ActiveX Data Objects (ADO) to display product information.

Module 4, “Managing A Shopping Cart,” shows students how to manage a Shopping Cart and implement Upsell Promotion.
Module 5, “Processing Orders,” shows students how to create an order-processing pipeline using the plan template.

Module 6, “Checking Out,” shows student how to capture shopper information, computing order value, and creating a Scriptor component for computing tax.

Module 7, “Completing the Purchase Process,” shows students how to use the Purchase template of the Order Processing Pipeline and secure financial transactions.

Module 8, “Tracking Shopper Information,” shows students how to track shoppers using cookies and registration database.

Module 9, “Implementing Business-to-Business E-Commerce,” shows students how to track orders from business partners.
Microsoft Certified Professional Program

Briefly describe the four certifications and the requirements for each of them. Refer to the student notes for supporting details.

The Microsoft Certified Professional (MCP) program provides the best method to prove your command of current Microsoft products and technologies. Anyone who must prove his or her technical expertise with Microsoft products should consider the program, including systems engineers, product developers, support technicians, system and network administrators, consultants, and trainers.

The Four Certifications

The following table describes the four certifications, which are based on specific areas of technical expertise.

<table>
<thead>
<tr>
<th>Certification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Certified Product Specialist (MCPS)</td>
<td>MCPSs demonstrate in-depth knowledge of at least one Microsoft operating system. Candidates may pass additional Microsoft certification exams to further qualify their skills with development tools, desktop applications, or the Microsoft BackOffice® family of products.</td>
</tr>
<tr>
<td>Microsoft Certified Systems Engineer (MCSE)</td>
<td>MCSEs are qualified to effectively plan, implement, maintain, and support information systems in a wide range of computing environments with Microsoft Windows NT® Server and Microsoft BackOffice.</td>
</tr>
<tr>
<td>Microsoft Certified Solution Developer (MCSD)</td>
<td>MCSDs are qualified to design and develop custom business solutions with Microsoft development tools, technologies, and platforms, including Microsoft Office and Microsoft BackOffice.</td>
</tr>
<tr>
<td>Microsoft Certified Trainer (MCT)</td>
<td>MCTs are instructionally and technically qualified to deliver Microsoft Official Curriculum through Microsoft Authorized Technical Education Centers.</td>
</tr>
</tbody>
</table>
Certification Requirements

The certification requirements differ for each certification and are specific to the products and job functions addressed by the certification. To become a Microsoft Certified Professional, you must pass rigorous certification exams that provide a valid and reliable measure of technical proficiency and expertise.

The following table describes exam requirements.

<table>
<thead>
<tr>
<th>Certification</th>
<th>Exam requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Certified Product Specialist (MCPS)</td>
<td>Pass one operating system exam. In addition, individuals seeking to validate their expertise in a desktop program must pass the appropriate elective exam.</td>
</tr>
<tr>
<td>Microsoft Certified Systems Engineer (MCSE)</td>
<td>Pass four operating system exams and two elective exams.</td>
</tr>
<tr>
<td>Microsoft Certified Solution Developer (MCSD)</td>
<td>Pass two core technology exams and two elective exams.</td>
</tr>
<tr>
<td>Microsoft Certified Trainer (MCT)</td>
<td>Requirements are specific to each Microsoft Official Curriculum course they are certified to deliver. Required to meet instructional and technical requirements.</td>
</tr>
</tbody>
</table>

For More Information  See the “Certification” section of the Web page provided on the compact disc or the Microsoft Training & Certification Web site at http://www.microsoft.com/train_cert/
MCSD Track

**Slide Objective**
To describe the recommended path to certification for the MCSE Windows NT Server 4.0 track.

**Lead-in**
If you are interested in becoming an MCSE, this is the path that we recommend.

Briefly describe each step.

Tell students that the Microsoft Training and Certification Web site will provide them with a complete list of all exams.

Tell students that this course is not part of the MCSE track, but it does have an online skills assessment.

Refer them to the Direct Access Web site for more information on the online skills assessment.

The following table outlines the recommended path to certification.

<table>
<thead>
<tr>
<th>Step</th>
<th>Pass these exams</th>
<th>Preparation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Desktop Applications Development (1 required):</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exam 70-016: Designing and Implementing Desktop Applications with Microsoft® Visual C++® 6.0</td>
<td>Course 1015: Mastering MFC Development Using Microsoft Visual C++ 6</td>
</tr>
<tr>
<td></td>
<td>Exam 70-176: Designing and Implementing Desktop Applications with Microsoft® Visual Basic® 6.0</td>
<td>Course 1013: Mastering Microsoft Visual Basic 6 Development</td>
</tr>
<tr>
<td>2</td>
<td>Distributed Applications Development (1 required):</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exam 70-015: Designing and Implementing Distributed Applications with Microsoft® Visual C++® 6.0</td>
<td>Course No. 1015 Mastering MFC Development Using Microsoft Visual C++ 6</td>
</tr>
<tr>
<td></td>
<td>Exam 70-175: Designing and Implementing Distributed Applications with Microsoft® Visual Basic® 6.0</td>
<td>Course No. 1298 Mastering Distributed Application Design and Development Using Microsoft Visual Studio 6.0</td>
</tr>
</tbody>
</table>

Pass Exam 70-100 Analyzing Requirements and Defining Solution Architectures

Pass One Elective Exam
<table>
<thead>
<tr>
<th>Step</th>
<th>Pass these exams</th>
<th>Preparation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Solution Architecture (required):</td>
<td>Course No. 1298 Mastering Distributed Application Design and Development Using Microsoft Visual Studio 6.0</td>
</tr>
<tr>
<td></td>
<td>Exam 70-100: Analyzing Requirements and Defining Solution Architectures</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Elective Exams (1 required):</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exam 70-019: Designing and Implementing Data Warehouses with Microsoft® SQL Server™ 7.0</td>
<td>Course No. 1502: Designing and Implementing a Data Warehouse Using Microsoft SQL Server 7.0</td>
</tr>
<tr>
<td></td>
<td>Exam 70-029: Designing and Implementing Databases with Microsoft® SQL Server™ 7.0</td>
<td>Course No. 833 Implementing a Database on Microsoft SQL Server 7.0</td>
</tr>
<tr>
<td></td>
<td>Exam 70-024: Developing Applications with C++ Using the Microsoft® Foundation Class Library</td>
<td>Course No. 789 Mastering MFC Development Using Microsoft Visual C++ 5</td>
</tr>
<tr>
<td></td>
<td>Exam 70-025: Implementing OLE in Microsoft® Foundation Class Applications</td>
<td>Course No. 789 Mastering MFC Development Using Microsoft Visual C++ 5</td>
</tr>
<tr>
<td></td>
<td>Exam 70-152: Designing and Implementing Web Solutions with Microsoft® Visual InterDev™ 6.0</td>
<td>Course No. 1017 Mastering Web Application Development Using Microsoft Visual InterDev 6</td>
</tr>
<tr>
<td></td>
<td>Exam 70-055: Designing and Implementing Web Sites with Microsoft® FrontPage® 98</td>
<td>Course No. 1009 Mastering Web Site Fundamentals</td>
</tr>
<tr>
<td></td>
<td>Exam 70-057: Designing and Implementing Commerce Solutions with Microsoft® Site Server 3.0, Commerce Edition</td>
<td>Course in development</td>
</tr>
</tbody>
</table>
Exam Preparation Guides

Microsoft Official Curriculum (MOC) helps you to prepare for Microsoft Certified Professional (MCP) exams. However, no one-to-one correlation exists between MOC courses and MCP exams.

To help prepare for the MCP exams, you can use the preparation guides that are available for each exam. Each Exam Preparation Guide contains exam-specific information, such as a list of the topics on which you will be tested and information about credit toward certification. These guides are available on the Training and Certification Web site, located at the following address:

http://www.microsoft.com/train_cert/

The Exam Preparation Guide for each exam is located on the page that discusses the status of that particular exam. To locate the status page for a particular exam from anywhere on the train_cert Web page, click Find Exam.
Facilities

Slide Objective
To inform students of class logistics and rules for the training site.

Lead-in
Before we start, let's go over the class logistics.

Explain the class hours, extended building hours for labs, parking, rest room location, meals, phones, message posting, and where smoking is or isn't allowed.

Also make sure that the students are aware of the recycling program if one is available.
Instructor Notes Module 1: Understanding Electronic Commerce

Introduction

This module provides students with the knowledge necessary to understand the importance of electronic commerce as an emerging business requirement. The module shows how customers buy products over the Web and defines Commerce Server as the tool that the students will use to create their own site.

After completing this module, students will be able to:

- Explain what electronic commerce (e-commerce) is.
- Describe the need for e-commerce.
- Explain how business is transacted on an e-commerce site.
- List the features and requirements of a good e-commerce site.
- List a few e-commerce sites on the Internet.
- Explain what Commerce Server is.
- List the Commerce Server tools that enable creation of customized e-commerce sites.

Materials and Preparation

This section provides you with the materials and preparation needed to teach this module.

Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file P01_1588.ppt
- Module 1, “Understanding Electronic Commerce”

Preparation

To prepare for this module, you should:

- Read all the materials for this module.
- Review the relevant sections of the product documentation.
- Browse through the sample site.
Module Strategy

Use the following strategy to present this module:

- **Overview of Electronic Commerce**
  Explain e-commerce, its applications and the need for building an electronic store. Discuss the advantages of opening an electronic store over setting up a physical store.

- **Tour of a Sample Site**
  Explain the following functionality while you demonstrate the sample site:
  a. The product catalog
  b. The shopping cart
  c. Price promotions, cross-sell promotions, upsell promotions
  d. Registration of shoppers
  e. Searching by various parameters
  f. Completing purchase of an item
  g. Tracking orders
  h. The business-to-business feature

  The implementation of the business-to-business feature on the Five Lakes Publishing sample site is as follows:
  
  Five Lakes Publishing advertises its presence on the Internet through other popular sites. The incentive for the owners of other Web sites to advertise the Five Lakes Publishing site is a commission on every sale referred through their sites. These Web-site owners are called trading partners or ePartners. Every ePartner is given a unique ID. In addition, every order has an ePartnerID associated with it. This enables Five Lakes Publishing to compute commissions for its trading partners.
  
  Five Lakes Publishing maintains a table that stores information about its trading partners.
  
  While demonstrating the business-to-business implementation, specify the epartner_id as 1, 2 or 3. The ePartnerID of 1 is assigned to Five Lakes Publishing.

- **Introduction to Commerce Server**
  Mention what functionality is automatically added by the Site Foundation Wizard and Site Builder Wizard, and the functionality that can be added by using the wizard-generated code.
  
  If time permits, encourage students to look at a few e-commerce sites such as www.dell.com and www.barnesandnoble.com.

**Recommended Reading**

- “*Understanding Electronic Commerce*” by David Koisur, published by Microsoft Press.
Module 1: Understanding Electronic Commerce
Overview

Objectives
After completing this module, you will be able to:

■ Explain what electronic commerce (e-commerce) is.
■ Describe the need for e-commerce.
■ Explain how business is transacted on an e-commerce site.
■ List the features and requirements of a good e-commerce site.
■ List a few e-commerce sites on the Internet.
■ Explain what Commerce Server is.
■ List the Commerce Server tools that enable creation of customized e-commerce sites.
◆ Overview of E-Commerce

**Slide Objective**
To provide an overview of the topics in this section.

**Lead-in**
This section introduces the key applications of, and the need for, e-commerce. You will also learn about the shopping process on an e-commerce site.

- What Is E-Commerce?
- Why Use E-Commerce?
- How Does E-Commerce Work?
What Is E-Commerce?

E-commerce (e-commerce) is a means of enabling and supporting the exchange of information, goods, and services between companies or between companies and their customers. It enables companies to be more efficient in their internal operations and more responsive to the needs and expectations of their customers. E-commerce technologies enable enterprises to exchange information instantaneously, eliminate paperwork, and advertise their products and services to a global market. E-commerce is divided into two categories: business-to-consumer and business-to-business commerce.

Business-to-consumer commerce

Business-to-consumer commerce is that in which businesses create electronic storefronts that offer information, goods, and services to consumers. There are now Internet “shopping malls” on the Web, which sell consumer goods ranging from cakes and wines to computers and cars.

An example of a business-to-consumer site is eToys.com.

Business-to-business commerce

Business-to-business commerce includes online wholesaling, in which businesses sell goods and services to other businesses on the Web. Business-to-business commerce is transacted by using Electronic Data Interchange (EDI) technologies. EDI defines the formats, data types, and routing instructions for the electronic exchange of business documents between companies’ computer systems.

An example of a business-to-business site is openmarket.com.
Why Use E-Commerce?

Slide Objective
To explain the need for building an e-commerce site.

Lead-in
Many Internet users are already buying goods such as computer software and hardware, books, CDs, and airplane tickets over the Internet. Other products and services will be entering the e-commerce arena soon.

- Lowers entry costs
- Reduces transaction costs
- Provides access to global market
- Provides for online distribution
- Secures market share

It is becoming increasingly important for businesses to use the Internet to reach new markets. This is because the Internet provides a flexible and dynamic marketplace for businesses to exchange goods, services, and information with consumers and business partners. The biggest advantage that being online offers a business is the ability to market products both locally and globally. The following are some reasons for companies to build a commerce-enabled Web site.

**Low entry costs** A company can establish itself on the Internet, and open for business, with a very small investment. There are thousands of companies with simple, inexpensive sites that are very successful in their markets.

**Reduced transaction costs** Dealing with customers over the Web, whether to process orders or to attend to customer support, is cheaper than traditional marketing methods. For example, Dell Computer Corporation estimates that they save eight dollars each time customers check the status of their order at the Dell Web site, instead of calling the company.

**Provides access to global market** With a traditional business, the target market may be the local community or, with a higher advertising budget, may extend to neighboring communities. The Web extends the reach of even the smallest businesses and allows them to market products globally.

**Provides for online distribution** The Web enables businesses to distribute data and software online.

**Secures market share** Getting a business online protects its current offline market share from being eroded by an online entrepreneur. If a business enters the e-commerce market too late, competitors who have already established their Web presence may make market entry more difficult.
How Does E-Commerce Work?

The following describes how the e-commerce process works.

1. A consumer uses a Web browser to connect to the home page of a merchant’s Web site on the Internet.
2. The consumer browses the catalog of products featured on the site and selects items to purchase. The selected items are placed in the electronic equivalent of a shopping cart.
3. When the consumer is ready to complete the purchase of selected items, they provide a bill-to and ship-to address for purchase and delivery.
4. When the merchant’s Web server receives this information, it computes the total cost of the order, including tax, shipping and handling charges, and then displays the total to the customer.
5. The consumer can now provide payment information, such as a credit card number, and then submit the order.
6. When the credit card number is validated, and the order is completed at the Commerce Server site, a receipt is displayed to the customer confirming the purchase.
7. The Commerce Server site then forwards the order to a Processing Network for payment processing and fulfillment.
Tour of a Sample Site

Slide Objective
To provide an overview of the topics in this section.

Lead-in
This section covers...

- Requirements of a Good E-Commerce Site
- A Few E-Commerce Sites on the Net
The following items are the critical requirements for businesses creating e-commerce solutions.

Promotional and searchable content
Companies online need to maintain product and service catalogs that display their offerings to shoppers in a clear and searchable format. Such catalogs should also promote special price and product offers.

Shopping cart
Online stores should provide “shopping carts” into which shoppers can place their selections. This product selection feature should allow shoppers to add or remove products from their cart and indicate quantity and per unit cost of the items in the cart.

Shopper management
Businesses need to engage their customers and business partners who are visiting and purchasing via their site. To do this, businesses can gather information about shopper interests and what products they buy. The next step is to maintain a shopper profile, with a customer’s purchase history and a record of their previous shopping carts’ contents. Finally, businesses can assist shoppers after they have bought a product by providing receipts, answers to frequently asked questions, and information on product improvements.

Flexible support for payment
Both businesses and consumers need e-commerce sites that offer secure and flexible, Internet-based transactions. Effective e-commerce solutions should support online payment through credit cards, as well as offline payment methods, such as check or cash.
Secure transactions

E-commerce sites should have security features that control the access of consumers, business trading partners, and operators to the site. Sites should also provide protection for shopper information, such as passwords and credit card numbers, transmitted over the Internet.
Demonstration: Visiting the Sample Site

In this demonstration, you will visit the sample site and observe the following functionality:

- The product catalog
- The shopping cart
- Price promotions, cross-sell promotions, upsell promotions
- Registration of shoppers
- Searching by various parameters
- Completing purchase of an item
- Tracking orders
- The business-to-business feature

**Note** You will be able to create a site similar to the sample site by adding the functionality listed above. To replicate the interface of the sample site, refer to Appendix A. You can modify the HTML code in the wizard-generated .asp (Active Server Pages) files to give your site the desired look and feel.

**To access the sample site**
1. Start Microsoft® Internet Explorer.
2. Type the URL address [http://Ecommerce/FiveLakes](http://Ecommerce/FiveLakes).

**To browse the product catalog**
2. View the available books under this section.
3. Click **Home** on the navigation bar to view books in a different section.
To shop at the sample site

2. On the Book Sections page, select Mastering Windows NT Server 4 from the list of available computer books by clicking it. Notice the cross-sell product on the Product Information page.
3. On the Mastering Windows NT Server 4 Product Information page, click BUY NOW to add this product to your shopping cart.
4. On the Cart page, click May we suggest a better book? to see the associated upsell product.
6. On the Cart page, click Continue Shopping to add more products to your cart.
8. On the Cart page, click Delete next to C++ Primer. Note that the discount on The C++ Programming Language is no longer applicable.
10. On the Shipping page, type in the shipping information, and then click Total.
11. On the Payment page, type in the credit card and billing information. You can use the following test credit card numbers:
   - American Express: 3111-1111-1111-1117
   - Visa: 4111-1111-1111-1111
   - MasterCard: 5111-1111-1111-1118
   - Discover: 6111-1111-1111-1116
12. Click PURCHASE to submit the order. Note the order number generated on the Purchase Confirmation page.

To track your order

1. On the Home page, click Order Status on the navigation bar.
2. On the Order Status page, type your order number in the Order Number text box, and then click Go.
3. Note the status of your order.

To search for products

1. On the Search page, select Category from the Search By drop-down list box.
2. In the Value text box, type Databases, and then click Go.
3. Note the search results.
To display orders received from business partners

1. Access the sample site by typing the URL address
2. Shop at the sample site and complete the purchase of a product.
3. Access the Manager page by typing the URL address
   http://Ecommerce/FiveLakes/Manager.
4. On the Site Manager page, click Orders in the Transactions section.
5. On the Order Manager page, click Orders by ePartner.
6. View the report.
A Few E-Commerce Sites on the Net

Microsoft Press
The Microsoft Press® online store provides learning and training resources for all levels of PC users and professionals. This site offers books, CDs, self-paced training kits, and videos to accommodate different learning styles. Shoppers can browse through the catalog of books or use the search function to see a list of books by title, author, or subject. The site features “cross-sell” promotions, which suggest other items that a shopper might be interested in.

Dell Computers
The Dell Computers online store allows shoppers to configure, price, and purchase customized desktop PCs, portables, and servers over the Internet. The online store reduces order costs and generates customized pricing and reports for corporate customers. Once shoppers have placed an order, they can log onto the Dell site to find out the status of any order. Shoppers can use Dell’s Order Watch system, which automatically sends them an e-mail notification once their order has been shipped. The site also offers extensive service and support features such as self-diagnostic tools, downloadable basic input/output (BIOS) files, drivers and utilities, and a public discussion area.

Barnes and Noble
The Barnes and Noble online bookstore offers a large selection of book titles. The site has a powerful search engine for fast results, and is designed for easy navigation. Book descriptions, reviews, and excerpts are provided for all titles on the site. Shoppers can browse through specialty sections for gifts, magazines, out-of-print and rare books. The site also offers shoppers features like the option to send gift certificates or have their gifts wrapped.
Introduction to Commerce Server

- Overview of Commerce Server
- Commerce Server Platform Architecture
- Important Commerce Server Tools
Overview of Commerce Server

Commerce Server enables enterprises to build powerful and cost-effective solutions for engaging and transacting with customers and business partners online. The features and functionality of Commerce Server provide organizations with the ability to create an e-commerce site quickly and easily. Commerce Server supports business-to-consumer as well as business-to-business transactions.

- Business-to-consumer sites: Commerce Server provides tools and features for building business-to-consumer sites, including support for the shopping cart feature, promotions, cross-sells, secure payment, and order processing.
- Business-to-business sites: Commerce Server provides tools and features for building business-to-business sites, including support for purchase orders, order approval routing, and secure exchange of business information between trading partners.

Using Commerce Server, you can:

Create an e-commerce site

You can quickly create a fully functional e-commerce site using the Site Foundation Wizard and Site Builder Wizard. The Site Builder Wizard creates Active Server Pages (.asp) files based on the options that you choose. Sites generated by the wizard can be edited using Microsoft Visual InterDev™. The wizard creates a site that displays a catalog of your products, accepts orders, and integrates with your existing business systems. For more information, see “Module 2: Building a Site.”

Customize the e-commerce site

Commerce Server provides tools, COM objects and samples that you can use to customize the way a site processes orders, integrates with existing business systems, and communicates with trading partners.
**Manage the e-commerce site**

The Site Builder Wizard generates a set of Web pages that are used for performing management tasks such as adding and deleting products, modifying the department structure, offering sales and promotions, checking orders, and so on. Changes are automatically reflected on the site’s pages. Access to these Manager pages is restricted to the site operator and Microsoft Windows NT® user accounts authorized by the site operator.
Commerce Server Platform Architecture

Commerce Server runs under the Windows NT Server operating system, which supports the security and performance requirements of Commerce Server.

Windows NT Server includes an integrated Web server, the Internet Information Server (IIS) version 4.0 and Microsoft Windows NT Server version 4.0 to provide a reliable and comprehensive Internet Commerce platform.

Commerce Server requires an ORACLE or SQL-compliant database and a database driver compatible with ODBC version 3.5 for data storage. A single Commerce Server site can have one or more databases, which can reside either on the same computer as the Commerce Server, or on a separate computer connected by Local Area Network (LAN).

A Web browser is required to connect to a commerce site of this type.
Important Commerce Server Tools

Commerce Server includes a number of tools that make it easier to create a new or a customized Commerce Server site. These tools include the Site Foundation Wizard, Site Builder Wizard, Pipeline Editor, and the Server Administration pages.

Site Foundation Wizard
The Site Foundation Wizard sets up the physical infrastructure of the site, including the directory structure for a new e-commerce site, virtual directories, site configuration files, Windows NT user account, and so on. After setting up the site foundation, the wizard generates a link to the new site’s main Site Manager page. This page provides access to the Site Builder Wizard.

Site Builder Wizard
The Site Builder Wizard provides a simple step-by-step interface that generates a fully functional e-commerce site quickly and easily. The wizard supports a variety of site options, including department hierarchies, products in multiple departments, variable numbers of product attributes, different registration models, price and cross-sell promotions, Microsoft Wallet payment options, and so on.

Pipeline Editor
The Pipeline Editor is used to customize the Order Processing Pipeline (OPP) of a site for complete integration with pre-existing business systems. The OPP provides a series of processing stages defining the method in which orders are routed through the site. Each OPP stage consists of one or more pipeline components designed to perform some operation on the business object. Developers can add, move, and reconfigure the OPP components in the order suited to their commerce application.
Server Administration pages

Server Administration pages are used for both local and remote administration of the server computer on which the e-commerce sites are hosted. Using this tool, the server administrator can access the Site Manager pages, open and close sites, create new sites, delete obsolete sites, and modify the properties of a site.
1. List 5 important requirements of a good e-commerce site.

   Provide promotional and searchable content, flexible support for payment, secure transactions, shopping cart, and shopper management.

2. What is the role of IIS in Commerce Server platform architecture?

   IIS provides secure Web site administration for Internet sites using Commerce Server. IIS also provides the ASP environment for developing Web-based applications and ActiveX Data Objects for dynamic access to the database from Commerce Server sites.

3. What can you do with Commerce Server?

   Using Commerce Server you can create, customize, and manage an e-commerce site.

4. Which Commerce Server tool creates the physical directory structure of a new site?

   The Site Foundation Wizard
Instructor Notes Module 2: Building a Site

Introduction

Presentation: 40 Minutes
Lab: 30 Minutes

This module provides students with the knowledge necessary to create a new Commerce Server site.

After completing this module, students will be able to:

- Create a foundation for a site by using the Site Foundation Wizard.
- Build the site by using the Site Builder Wizard and populate it with departments, products and product attributes.
- Perform simple administrative tasks by using the Server Administration pages.

Materials and Preparation

This section provides you with the materials and preparation needed to teach this module.

Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file P02_1588.ppt
- Module 2, “Building a Site”
- Lab 2, “Creating a New Commerce Server Site”
- Classroom Setup Guide file S01_1588.doc

Preparation

To prepare for this module, you should:

- Read all the materials for this module.
- Complete the lab.
- Review the relevant sections of the product documentation.
- Read through the classroom setup guidelines.
Instructor Notes Module 2: Building a Site

Instructor Setup for a Demonstration

Demonstration: Using the Site Foundation Wizard
Ensure that a System DSN named DemoStore has been created on the instructor and student machines.

Instructor Setup for a Lab

Lab 2: Creating a New Commerce Server Site
Ensure that the FiveLakes database and the DSN named FiveLakes exist on all student machines.

Module Strategy

Use the following strategy to present this module:

- Creating a Site Foundation
  When demonstrating the Site Foundation Wizard, mention that the wizard can be started either using the Server Administration pages or the Microsoft Management Console (MMC).
  Explain the options on each screen as you demonstrate the Site Foundation Wizard.

- Creating a Site
  Explain the options on each screen as you demonstrate the Site Builder Wizard.
  Mention that once a site is created, the Server Administration pages are used to open, close, delete, or create sites, apart from modifying the properties of a site.

- Working with the Server Administration Pages
  Demonstrate the options in the Server Administration pages.
  Wherever possible compare how tasks are accomplished in the Server Administration pages to how they are done in the MMC.
Module 2: Building a Site
Overview

Objectives
After completing this module, you will be able to:

- Create a foundation for a site by using the Site Foundation Wizard.
- Build the site by using the Site Builder Wizard and populate it with departments, products, and product attributes.
- Perform simple administrative tasks by using the Server Administration pages.
Creating a Site Foundation

Slide Objective
To provide an overview of the topics in this section.

Lead-in
In this section, you will learn to use the Site Foundation Wizard.

- What the Site Foundation Wizard Creates
- Prepare a Database for the Site
- Demonstration: Using the Site Foundation Wizard
What the Site Foundation Wizard Creates

The foundation for a site is created by using the Site Foundation Wizard. The foundation created by the wizard includes:

- The physical directory structure of the new site.
- The virtual directory.
- The site configuration files:
  - Site_name\Config\Site.csc for the site pages.
  - Site_name\Manager\Config\Site.csc for the Site Manager pages.
- The URL to Site Manager page.

Among other settings, the site configuration files store the connection string for the new site’s database and the name of the Microsoft® Windows NT® account that has permission to access the site’s Manager pages.

When the Site Foundation Wizard has created the foundation, it displays the URL for the new site’s Manager page. At this point, the Manager page is blank except for a link that starts the Site Builder Wizard. Using the URL, the site operator can connect to the Manager page and run the Site Builder Wizard, which generates the Active Server Pages (ASP) files for the new site and the site’s Manager pages.
Prepare a Database for the Site

- Create the site database
- Create database logins
- Create a Data Source Name (DSN)

Creating the site database

The first step in preparing a database for the site is to create the database itself. This database will store information about the products offered at your site and the transactions that occur on your site.

To create a new database in SQL Server 7.0, you must determine the name of the database, its owner, its size, and the files that will be used to store it. Every database has one primary file that stores data, and a transaction log that stores the log information used to recover the database. You can create databases using Transact-SQL, SQL Server Enterprise Manager, the Create Database Wizard, or by using SQL-DMO programmatically. For more information, see Microsoft SQL Server 7.0 product documentation.

Creating database logins

The next step is to add login names to the SQL Server for the site operators and site visitors. You can create login names by using SQL Server Enterprise Manager. The site operator’s login needs to have full permissions on the database as the Site Builder Wizard uses this login name when it creates the database schema for the new site. The site visitor’s login name has a restricted set of permissions since the site visitor uses this login only to shop at the site.

Creating a Data Source Name (DSN)

The last step is to create a Data Source Name by using the Microsoft Open Database Connectivity (ODBC) Data Source Administrator utility in Control Panel.

When you run the Site Foundation Wizard, you must provide a DSN, a database login name and password that enables the wizard to construct a connection string, used to connect to the new site’s database.
Demonstration: Using the Site Foundation Wizard

In this demonstration, you will use the Site Foundation Wizard to create a foundation for a new site.

To access the Site Foundation Wizard
1. Click the Start button, point to Programs, Microsoft Site Server, Administration, and then click Site Server Service Admin (HTML).
2. On the Site Server Web Administration page, click Commerce.
3. On the Getting Started with Site Server Commerce page, click Server Administration.
4. On the Server Administration page, click Create.

To create a site foundation with the Create New Site Foundation Wizard
1. On the Create New Site Foundation Wizard - Select a Web Site page, select Default Web Site, and then click Next.
2. On the Select a Site Name page, type DemoStore into the Short name and Demo Book Store into the Display name text boxes respectively, and then click Next.
3. On the Select a Directory Location page, click Next to accept the default directory.
4. On the Formulate a Database Connection String page, select the DSN DemoStore in the Available DSNs list box. Type sa in the Database login text box, leave the Database password text box blank, and then click Next.
5. On the Specify Manager Account page, click Next to use an existing account.
6. On the Select Windows NT Domain page, select the domain for your local computer, and then click **Next**.

7. On the Select a Windows NT account page, select **Administrator**, and then click **Next**.

8. On the Finish page, click **Finish**.

   Leave the wizard in this state. The next demonstration picks up from here.
◆ Creating a Site

**Slide Objective**
To provide an overview of the topics provided in this section.

**Lead-in**
In this section, you will learn to create and test a site.

- What the Site Builder Wizard Generates
- Demonstration: Creating and Testing a Site
What the Site Builder Wizard Generates

After the foundation for the new site has been created, the site operator can connect to the Manager page and run the Site Builder Wizard. Based on the options selected, the Site Builder Wizard generates the ASP files, database tables and schema for the new site and its corresponding Site Manager pages. If required, it can also generate sample data and load it into the database. The wizard prefixes the site name to each table name. For example, if the site name is specified as Test, the new basket table is named Test_basket.

The following table lists and describes the default tables generated by the Site Builder Wizard.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sitename_basket</td>
<td>Contains information about the items that a shopper has selected during a shopping session.</td>
</tr>
<tr>
<td>Sitename_dept</td>
<td>Contains information about the various departments on the site.</td>
</tr>
<tr>
<td>Sitename_dept_prod</td>
<td>Associates each product to a specific department.</td>
</tr>
<tr>
<td>Sitename_product</td>
<td>Contains information on each product.</td>
</tr>
<tr>
<td>Sitename_promo_cross</td>
<td>Contains information about the products for which cross-sell products exist.</td>
</tr>
<tr>
<td>Sitename_promo_price</td>
<td>Contains information about the price discounts offered in a site.</td>
</tr>
<tr>
<td>Sitename_receipt</td>
<td>Contains order information with order number as the primary key.</td>
</tr>
<tr>
<td>Sitename_receipt_item</td>
<td>Associates each item to a specific order.</td>
</tr>
<tr>
<td>Sitename_shopper</td>
<td>Contains personal details of visiting shoppers.</td>
</tr>
</tbody>
</table>
Demonstration: Creating and Testing a Site

Slide Objective
To demonstrate the creation of a new Commerce Server site.

Lead-in
In this demonstration, you will build a new Commerce Server site and test it by shopping at this site.

In this demonstration, you will:

- Build a site on the foundation by using Site Builder Wizard.
- Populate the site with a department and a product.
- Test the store by shopping for the product created.

► To access the Site Builder Wizard
1. On the Create New Site Foundation Wizard - Site Creation Complete page, click http://localhost:80/DemoStore/manager/default.asp, which is the URL to the new site’s Manager page.
2. On the new site’s Manager page, click Commerce Site Builder Wizard.

► To create a site on the foundation with the Site Builder Wizard
1. On the Welcome page, click Next.
2. On the Site Type page, click Next to create a custom site.
3. On the Merchant Information page, type information about the site you are building, and then click Next.
4. On the Locale page, click Next to accept the default of English (United States).
   The locale configures the new site to calculate taxes and to display currency, time, and address in the proper format.
5. On the Site Style page, set the navigation bar, font, background color, and button style to your preference, and then click Next.
6. On the Promotions page, select the Price promotions and Cross-sell promotions check boxes, and then click Next.
7. On the Features page, under Registration, accept the default None, and under Department Type, accept the default Simple. Make sure that under Product Searching, Enabled is selected, and then click Next.

8. On the Product Attribute Type page, click Next to accept Static Attributes.

9. On the Product Structure page, add Author as a single valued custom field, and click Next.

10. On the Shipping & Handling page, select the Enabled check box for Handling charges, type in $2.00 as cost for handling, and then click Next.

11. On the Tax: USA page, click Enabled for a tax on Georgia (GA) residents only. In the Rate box, type 5.0, and then click Next.

12. On the Payment Methods page, click Next to accept the default credit cards.

13. On the Order History page, click the Retain order history and receipt information check box, and then click Next.

14. On the Output Options page, click Enable next to the Load Schema into Database option, clear the Generate Sample Data to file option, and then click Finish to create your site.

15. When the word “Done” appears at the bottom of the Shopper Site Pages list, click Here is your manager site.

▶ To add a department and product to the new site

**Note** If an unexpected HTTP application restart screen appears, click Refresh to clear it.

1. On the Site Manager page, in the Merchandising section, click Departments.

2. On the Departments page, click Add New Department.

3. On the New Department page, in the Dept Id field, type 1, in the Name field, type Computers, in the Description field, type The World of Computers, and then click Add Department.

4. On the Departments page, notice that the first department has been created, and then click Products.

5. On the Products page, click Add New Product.

6. On the New Product page, type in the following parameters:
   a. Sku = 001
   b. Name = Windows NT 4.0 MCSE Study Guide
   c. Description = Just the minimum needed
   d. List price = $62.99
   e. Author = Alan R.Carter, Thomas C. Willingham

7. Click Add Product.

8. On the Products page, click Departments.

9. On the Departments page, click Manager.

10. On the Site Manager page, in the System section, click Shop Site.
To test the store by shopping for the product that you have added

1. On the opening page of the store, click the **Computers** department.
2. On the Computers department page, click **Windows NT 4.0 MCSE Study Guide**.
3. On the Product Information page, click **Add to Basket** to add this book to the shopping cart.
4. On the Shopping Basket page, click **Purchase**.
5. On the Shipping page, select **Overnight** as the shipping method, type in the ship-to information, and then click **Total**.
6. If a Security Information box appears, click **Yes** to release the shipping address to the Web.
7. On the Final Purchase Approval page, type in the credit card information and the bill-to information. You can use the following test credit card numbers:
   - American Express: 3111-1111-1111-1117
   - Visa: 4111-1111-1111-1111
   - MasterCard: 5111-1111-1111-1118
   - Discover: 6111-1111-1111-1116
8. Click **Purchase** to submit the order.
9. On the Purchase Confirmation page, click on the order number to view your receipt.
The Server Administration pages are used to perform the following administrative tasks.

- Create new Commerce Server sites.
- Open, close, and delete Commerce Server sites.
- Modify the properties of a site, such as the database connection string of Commerce Server sites.
- Access the Site Manager pages.

**To access the Server Administration page**

1. Start Microsoft Internet Explorer.
2. Type the URL [http://localhost/siteserver/admin/commerce](http://localhost/siteserver/admin/commerce) into the Internet Explorer address bar.
3. On the Getting Started with Site Server Commerce page, click **Server Administration**.

The following are displayed on the Server Administration page:

- A list of Commerce Server sites.
- The version of Commerce Server that was used to create the site.
- The site’s status as open, closed, or invalid.
- The name of the IIS virtual directory.
- The Web site on which the site is located.
Opening, Closing, and Deleting Sites

To open, close, or delete one or more Commerce Server sites

1. On the Server Administration page, select the site or sites in the list of Commerce Server sites.
2. Click the Open button, Close button, or the Delete button.

On clicking Delete, a confirmation page is displayed on which you can specify whether you want to delete the site’s files and database tables in addition to the site’s virtual directory.
Modifying the Properties of a Site

Using the Server Administration pages, you can modify the following properties of a site.

- Change the display name of a Commerce Server site.
- Add or remove a Windows NT user account.
- Add, remove, or edit named connections in the connection map.

A connection string consists of a DSN, a database login name, and a database password. You can create a name to represent each connection string, and then store these named connections in the site’s connection map.

- Specify security options for a Commerce Server site, such as the URL for the page that should be displayed when the site is closed.
Managing a Site

Using the Site Manager page, you can:

- Add departments and products to your site.
- Edit a pipeline.
  
  For more information, see “Module 6: Checking Out.”
- Add price and cross-sell promotions.
- Open and close a site.
- Reload a site.

Some modifications that you make to a site, such as changes to its connection strings, are not reflected in the site until you reload it. Reloading the site causes IIS to rerun the site’s global.asa file, which picks up any changes.
Lab 2: Creating a New Commerce Server Site

After completing this lab, you will be able to:

- Create a site foundation by using the Site Foundation Wizard.
- Build a site by using the Site Builder Wizard.
- Test the store by shopping at its site.
1. What are the names of the site configuration files, and what kind of configuration data do they contain?

   The site configuration file for the site pages is Site_name\Config\Site.csc and the file for the Site Manager pages is Site_name\Manager\Config\Site.csc. The site configuration files store the connection string for a site’s database and the name of the Windows NT account that has permission to access the site’s Manager pages.

2. What is the best way to test a newly created site?

   The best way to test a site setup is to enter the site as a shopper.

3. What does the Site Builder Wizard do?

   The Site Builder Wizard generates the ASP files, database schema, and tables for the new site as well as its corresponding Site Manager pages.

4. What does the database table Sitename_basket contain?

   The Sitename_basket table contains information about the items that a shopper has selected during a shopping session.
Instructor Notes Module 3: Enhancing the Product Catalog

Introduction

Presentation: 60 Minutes
Lab: 30 Minutes

This module provides students with the knowledge necessary to enhance product catalog pages of a Commerce Server site by displaying custom attributes and cross-sell items on the Product page.

After completing this module, students will be able to:

- Describe Commerce Server objects.
- Modify product catalog pages generated by the Site Builder Wizard to display custom attributes.
- Add product search capability to the site.
- Implement Cross-sell functionality in the site.

Materials and Preparation

This section provides you with the materials and preparation needed to teach this module.

Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file P03_1588.ppt
- Module 3, “Enhancing the Product Catalog”
- Lab 3, “Customizing Product Catalog Pages”

Preparation

To prepare for this module, you should:

- View the relevant .asp files of the Five Lakes Publishing sample site.
- View the relevant Web pages of the Five Lakes Publishing sample site.
- Complete the practice and lab in Module 3.
Module Strategy

Use the following strategy to present this module:

- **Introducing Commerce Server Objects**
  
  Describe the services provided by Commerce Server objects. Explain the frequently used **Page** object methods with examples from the Five Lakes Publishing sample site. Describe the code in global.asa to explain how a **QueryMap** object is implemented.

- **Modifying Wizard-Generated Catalog Pages**
  
  Describe the elements generated by the Site Builder Wizard for catalog presentation. Encourage students to think of different styles of catalog presentation. Describe the code in global.asa and product.asp to add custom attributes to the Product page.

- **Adding Product Search Capability**
  
  Explain how search capabilities on a site can provide a quick and easy way of browsing the catalog.

- **Implementing Cross-Sell**
  
  Explain how Cross-sell will encourage shoppers to look at other related products on the site. Do not discuss Upsell here. Describe the code in basket.asp that retrieves cross-sell items from the database for each item in the shopping cart.
Module 3: Enhancing the Product Catalog
Overview

Slide Objective
To provide an overview of the module topics and objectives.

Lead-in
In this module, you will use the Commerce Server objects in ASP pages. You will also implement Cross-sell using the Site Manager pages.

Objectives

After completing this module, you will be able to:

- Describe Commerce Server objects.
- Modify product catalog pages generated by the Site Builder Wizard to display custom attributes.
- Add product search capability to the site.
- Implement Cross-sell functionality in the site.
Introducing Commerce Server Objects

Slide Objective
To provide an overview of the topics in this section.

Lead-in
Commerce Server objects support methods and properties that provide an extensive set of services.

- Commerce Server Objects
  - The Page Object
  - The QueryMap Object
- Understanding the Global.asa File

Installing Commerce Server registers Commerce Server objects on a system, which are Microsoft® ActiveX® Server objects. These objects provide an extensive set of services, including:

- Storing customer and purchase information for the current shopping session.
- Storing receipts and order information in a database.
- Defining queries to retrieve information from a site database.
- Formatting HTML page items.
- Retrieving values from a URL query string and converting the values to specific data types.

Commerce Server objects support methods and properties that provide these services. On instantiating a Commerce Server object, you can call its methods and properties from within Microsoft Visual Basic® Scripting Edition or Microsoft Jscript® server-side code that runs on a Commerce Server site.

The following table lists and describes a few Commerce Server objects used in a Commerce Server site.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>Provides methods to format HTML page items, generate URL strings, retrieve values from a URL query string and convert the values to specific data types.</td>
</tr>
<tr>
<td>QueryMap</td>
<td>Contains multiple query descriptions to retrieve information from a site database.</td>
</tr>
<tr>
<td>Object</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>OrderForm</strong></td>
<td>Supports the in-memory storage of shopper and purchase information for the current shopping session. For more information on the <strong>OrderForm</strong> object, see “Module 4: Managing a Shopping Cart.”</td>
</tr>
<tr>
<td><strong>Dictionary</strong></td>
<td>Provides for the in-memory storage of name/value pairs.</td>
</tr>
<tr>
<td><strong>SimpleList</strong></td>
<td>General-purpose list of variants.</td>
</tr>
<tr>
<td><strong>DBStorage</strong></td>
<td>Supports mapping <strong>Dictionary</strong> and <strong>SimpleList</strong> objects to and from a database, for the storage and retrieval of order information. For more information on the <strong>DBStorage</strong> object, see “Module 4: Managing a Shopping Cart.”</td>
</tr>
<tr>
<td><strong>StandardSMManager</strong></td>
<td>Facilitates the creation, deletion, and retrieval of shopper IDs. For more information on the <strong>StandardSMManager</strong> object, see “Module 4: Managing a Shopping Cart.”</td>
</tr>
</tbody>
</table>
The Page Object

The Page object is a Commerce Server object that is usually created on every ASP page displayed in the shopping process. The Site Builder Wizard in Commerce Server provides the following definition of a Page object in i_shop.asp, which is a wizard-generated .asp file.

```vbscript
```

The Page object supports methods that make it possible to easily format HTML page items, generate URL strings, retrieve values from a URL query string and convert the values to given data types based on a locale value.
The following table lists the frequently used Page object methods in the .asp files of the Five Lakes Publishing sample site.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| HTMLEncode   | Applies HTML encoding to a specified text string. Characters in the string such as “<” and “&” that have special meanings in HTML are converted to their HTML equivalents, such as &lt; and &amp; so that they are displayed correctly by a client browser. | The following code in dept.asp invokes the mscsPage.HTMLEncode method to apply HTML encoding to the department name so that it can be displayed correctly by the shopper’s browser:  
```html
<title>% = displayname%:Department: % = mscsPage.HTMLEncode (dept_name)%; </title>
``` |
| GetShopperID | Returns the shopper ID for the current shopper from a URL query string or a cookie, depending on the initialization mode of a Commerce Server site.                                                                 | The following code in i_shop.asp file, which is included in basket.asp, retrieves the shopper ID from the URL. The script in basket.asp uses the shopper ID to extract the shopper’s cart items from the database.  
```javascript
mscsShopperID = mscsPage.GetShopperID
``` |
| PutShopperID | Writes the specified shopper ID to a cookie or stores it in a URL, depending on the initialization mode of a Commerce Server site.                                                                 | The following code in i_shop.asp file, which is included in basket.asp, stores the specified shopper ID in the URL. The script in basket.asp uses the shopper ID to associate every item in the cart with the specified shopper ID.  
```javascript
mscsPage.PutShopperID (mscsShopperID)
``` |
| RequestString| Retrieves a value from a URL query string or form post variable and based on the specified locale, processes it. This processing may involve removing carriage returns, removing leading and trailing spaces, checking the string length against a specified range, and validating the string based on the specified or default locale. | The following code in product.asp retrieves a Product ID from the URL. The script in product.asp uses the Product ID to retrieve information about a product from the database.  
```javascript
sku = mscsPage.RequestString ("sku")
``` |
| RequestNumber| Retrieves a value from a URL query string or form post variable and converts it to a number based on the specified or default locale. Optionally, RequestNumber checks the converted number against the specified range, and validates it based on the specified locale. | The following code in dept.asp retrieves the department ID from the URL. The script in dept.asp uses the department ID to display products in the department.  
```javascript
dept_id = mscsPage.RequestNumber ("dept_id","0")
``` |
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
<td>Generates an OPTION item in an HTML form selection list, and assigns it a value and selection state.</td>
<td>The <code>mscsPage.Option</code> method in <code>payment.asp</code> generates a HTML selection field to enable a shopper to select the expiration year of the credit card. For the implementation of the <strong>Option</strong> method, see <code>payment.asp</code> file of the Five Lakes Publishing sample site in Appendix A.</td>
</tr>
</tbody>
</table>
The QueryMap Object

Creating a QueryMap object

Commerce Server sites use database queries to retrieve information from the database. Query statements are collectively stored in a QueryMap. Each query in the QueryMap is specified by means of a query description.

A query description includes at least one property called SQLCommand. The SQLCommand property is set to a SQL statement that will retrieve information from the site database. For more information on defining SQL statements, see Microsoft SQL Server 7.0 Books Online.

A QueryMap object is created in the Application_OnStart subroutine in the site’s global.asa file (configuration file for each Commerce site) as follows:

```vbscript
set MSCSQueryMap = Server.CreateObject
("Commerce.Dictionary")
```

Adding queries to a QueryMap object

A query description is added to the MSCSQueryMap object as follows:

```vbscript
set MSCSQueryMap.departments = Server.CreateObject
("Commerce.Dictionary")
```

Next, the SQLCommand property of departments is set to a SQL statement as follows:

```vbscript
set MSCSQueryMap.departments.SQLCommand = “select dept_id, dept_name from <Sitename>_dept where dept_id = :I”
```

where : ”I” in the SQL statement serves as a parameter that can be replaced with a value when the QueryMap object is used.
For more information on how query descriptions are added to a **QueryMap** object in the global.asa file of the Five Lakes Publishing sample site, see Appendix A.
Each Commerce Server site contains a global.asa, a configuration file that stores event scripts and objects used by the application. The objects are created as Application objects in global.asa, so that they can be accessed from within any page in the site.

In global.asa, the following objects are created and initialized.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileDocument</td>
<td>Reads and writes configuration information to and from the site configuration file (site.csc) and the Site Dictionary.</td>
</tr>
<tr>
<td>Site Dictionary</td>
<td>Contains the configuration data.</td>
</tr>
<tr>
<td>QueryMap Dictionary</td>
<td>Contains description of SQL queries used by the site.</td>
</tr>
<tr>
<td>MessageManager</td>
<td>Stores messages used by pipeline components.</td>
</tr>
<tr>
<td>StandardSManager</td>
<td>Initializes the application mode to cookie, url, cookieurl or urlcookie.</td>
</tr>
<tr>
<td>DataFunctions</td>
<td>Formats and converts values based on a specific locale, as well as performs validation checks on values passed to the pipeline or saved to the database.</td>
</tr>
</tbody>
</table>

For more information on how objects are created in the global.asa file of the Five Lakes Publishing sample site, see Appendix A.
Modifying Wizard-Generated Catalog Pages

- What the Wizard Generates
- Customizing Product Catalog Pages

Commerce Server sites are designed to contain catalog pages that display departments and their products. Catalog pages are created as standard Active Server Pages (ASP) files that run queries on the product database using Microsoft ActiveX Data Objects (ADO).

For more information on Active Server Pages, see Introducing Active Server Pages in the Product Documentation of Windows NT® 4.0 Option Pack.
The following table describes the elements created and included by the Site Builder Wizard for catalog presentation.

<table>
<thead>
<tr>
<th>Element</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Department table</td>
<td>Site database</td>
<td>Contains information on the various departments present in your site.</td>
</tr>
<tr>
<td>Department-Product table</td>
<td>Site database</td>
<td>Contains information, which associates each product to a specific department.</td>
</tr>
<tr>
<td>Product table</td>
<td>Site database</td>
<td>Contains information on the various products present in your site.</td>
</tr>
<tr>
<td>Query descriptions</td>
<td>Global.asa</td>
<td>Defines queries to extract department and product information from the site database.</td>
</tr>
<tr>
<td>Scripts in .asp files</td>
<td>Default.asp, dept.asp and product.asp</td>
<td>Contains code to execute queries and display the catalog.</td>
</tr>
</tbody>
</table>
Customizing Product Catalog Pages

**Slide Objective**
To display custom attributes on the catalog page.

**Lead-in**
The wizard-generated files need to be modified to change the default appearance of the catalog.

**Displaying custom attributes on the catalog page**
- Modify the query description in global.asa
- Modify the script in product.asp

Commerce Server allows you to modify wizard-generated catalog pages. For example, in the Five Lakes Publishing sample site, the default catalog generated by the wizard will display only the title, description, and price for each book. In order to display custom attributes such as author, edition, and publisher, you will need to modify:

- The query description `product_by_sku` in global.asa to include author, edition, and publisher attributes in the select statement as follows:

```sql
set MSCSQueryMap.product_by_sku = AddQuery "
(SELECT pf.sku, pf.name, pf.description, 
 pf.list_price, pf.sale_price, pf.sale_start, 
 pf.sale_end, pf.image_file, pf.image_width, 
 pf.image_height, pf.author, pf.edition, pf.publisher, 
 dept.dept_id, dept.dept_name 
 FROM <Sitename>_product pf, <Sitename>_dept_prod 
 deptprod, <Sitename>_dept dept 
 WHERE pf.sku = :1 AND pf.sku = deptprod.sku AND 
 dept.dept_id = deptprod.dept_id AND 
 dept.dept_id = :2")
```
The script in product.asp, to display author, edition, and publisher on the Product page along with other attributes as follows:

REM -- get author, edition, and publisher fields from recordset
author = rsProduct("author").value
edition = rsProduct("edition").value
publisher = rsProduct("publisher").value

REM - display the fields in HTML format
<P>%mscsPage.HTMLEncode(author) %>
<P>%mscsPage.HTMLEncode(edition) %>
<P>%mscsPage.HTMLEncode(publisher) %>
◆ Adding Product Search Capability

**Slide Objective**
To provide an overview of the topics in this section.

**Lead-in**
Product search capability provides a quick and easy method of looking for specific information.

- Adding Search Capabilities
- Practice: Implementing Search by Various Parameters
Adding Search Capabilities

As you saw in the previous module, the Site Builder Wizard provides an option to enable product search capabilities on your site. This enables shoppers to look for products in the site that match a specified keyword. The search page (find.asp) can be extensively tailored to the needs of your shoppers and to the information that is available on your catalog.

For example, to enhance the search capabilities of the Five Lakes Publishing sample site, you can allow shoppers to look for books by specifying an author, publisher, or book category.

The find.asp file can be modified to include:

- A list of search options to be displayed on the page.
- SQL queries with conditions based on the search option selected.
Practice: Implementing Search by Various Parameters

Slide Objective
To discuss the logic for implementing search on a site.

Lead-in
In this practice, you will add search capabilities to a site.

In this practice, you will add search options to your site that will enable shoppers to look for specific books.

► **Open an .asp file using Microsoft Visual InterDev 6.0**

1. On the **Start** menu, click **Programs, Microsoft Visual InterDev 6.0**.
2. On the New Project page, type **FiveLakes** as your project name, accept the default location, and then click **Open**.
3. In step 1 of the Web Project Wizard, enter **localhost** in the **What server do you want to use?** drop-down list box, select **Master mode**, and then click **Next**.
4. In step 2 of the Web Project Wizard, select the **Connect to an existing Web application on localhost** option button, select **FiveLakes** from the **Name** drop-down list box, and then click **Finish**.

   If you are asked, install the scripting library.

5. In the Project Explorer window, double-click **global.asa** to modify the code in this file.
Define query descriptions in global.asa

1. In the `InitQueryMap` function, add the following query descriptions to retrieve product information from the site database for a specific keyword or category.

   ```javascript
   set MSCSQueryMap.find_by_keyword = AddQuery
   ("SELECT p.sku, p.name, p.list_price, dp.dept_id FROM FiveLakes_product p, FiveLakes_dept_prod dp WHERE p.name like '%$:1%' AND p.sku = dp.sku order by p.name")
   
   set MSCSQueryMap.find_by_category = AddQuery
   ("SELECT p.sku, p.name, p.category, p.list_price, dp.dept_id FROM FiveLakes_product p, FiveLakes_dept_prod dp WHERE p.category = ':1' AND p.sku = dp.sku order by p.name")
   ``

2. On the File menu, click Save global.asa, and then click Close.

Display search options on the Search page

1. Double-click `find.asp` in the Project Explorer window.

2. Scroll to the form tag in find.asp and add the following code below the line that reads “Find:” to:
   a. Display a list of 2 search options – Keyword and Category.
   b. Store the selected option in a variable named `search_by` and the search value in a variable named `strFindSpec`.

   ```html
   <%
   search_by = trim(mscsPage.RequestString("sselect"))
   strFindSpec = mscsPage.RequestString("find_spec")
   %>
   <td width="125%" colspan="2"><select name="sselect" size="1" tabindex="20">
   <% if mscsPage.Option("Keyword", search_by) %>Keyword
   <% if mscsPage.Option("Category", search_by) %>Category
   </select></td>
   ```

Execute an appropriate query that will search the database

1. Scroll to the following line of code in find.asp that defines a query for retrieving product information for a specified keyword whose value is stored in a variable `safeFindSpec`.

   ```sql
   ```
2. Replace this line with the following code to check the content of the variable `search_by` and assign the appropriate query description value to a variable `sqlText` depending on its content:

   ```sql
   if search_by = "Keyword" then
       sqlText = MSCSQueryMap.find_by_keyword.SQLCommand
   elseif search_by = "Category" then
       sqlText = MSCSQueryMap.find_by_category.SQLCommand
   end if
   
   sqlText = Replace(sqlText, ":1", safeFindSpec)
   cmdTemp.CommandText = sqlText
   ```

1. On the **File** menu, click **Save find.asp**, and then click **Close**.

---

**Search for products on the site**

1. Start Microsoft Internet Explorer and type the URL
   http://localhost/FiveLakes in the address bar of Internet Explorer.
2. On the site Home page, click **Find**.
3. On the Find page, select **Category** from the drop-down list box next to **Find:**.
4. Type **Databases** in the text box provided for search value, and then click **Find**.
5. Notice the search results.
◆ Implementing Cross-Sell

**Slide Objective**
To provide an overview of the topics in this section.

**Lead-in**
Cross-sell promotions encourage shoppers to consider purchasing a related product.

- What the Site Builder Wizard Generates
- Using Promotions Wizard

Cross-sell promotions encourage shoppers to consider purchasing a related product. For example, in the Five Lakes Publishing sample site, the Product page for Essential Windows NT System Administration promotes Mastering Windows NT Server 4 as a related product.
What the Site Builder Wizard Generates

The Site Builder Wizard includes the following elements for Cross-sell promotions:

- A Cross-sell Promotions table (<Sitename>_promo_cross), which specifies which related product is to be cross-promoted.
- A script in the Product page (product.asp), which executes a query to retrieve cross-sell items. The script also displays the related products under the heading “See Also” on the Product page.

The Cross-Sell Promotions Table
The <Sitename>_promo_cross table contains 2 columns: Sku and RelatedSKU. Sku is the Product-ID for which related products exist. RelatedSKU is the Product-ID of the product that will appear under the heading “See Also” on the Product page.
The Product Page

The following script in product.asp retrieves information on related products from the Cross-sell Promotions table and displays it on the Product page.

```<%REM get related products (if any):
    cmdTemp.CommandText = Replace("SELECT prod.sku, prod.name, deptprod.dept_id FROM <Sitename>_promo_cross promo_cross, <Sitename>_product prod, demo_dept_prod deptprod WHERE promo_cross.sku = :1 and prod.sku = deptprod.sku AND and promo_cross.rel_sk = prod.sku", ":1", quoted_sku)
    Set rsRelated = Server.CreateObject("ADODB.Recordset")
    rsRelated.Open cmdTemp, , adOpenForwardOnly, adLockReadOnly

REM display up to 5 related products:
if Not rsRelated.EOF then
    <BR>
    <B>See Also</B>
    <%
    nRelated = 0
    set skuField = rsRelated("sku")
    set nameField = rsRelated("name")
    set dept_idField = rsRelated("dept_id")
    do while Not (rsRelated.EOF Or nRelated >= 5)
        <BR>
        <A HREF="<%= baseURL("product.asp") & mscsPage.URLShopperArgs("sku", skuField.value, "dept_id", dept_idField.value) %>" %>&</A>
        <%
        nRelated = nRelated + 1
        rsRelated.MoveNext
        loop
    <% end if %>
```
Using Promotions Wizard

Slide Objective
To list the steps to add, edit, and delete a Cross promotion.

Lead-in
The Promotions Wizard of Commerce Server enables you to add, edit, and delete related products.

The Promotions Wizard of Commerce Server enables you to add, edit, and delete related products on your site.

► To add related products to the Cross-sell Promotions table
1. Use your browser to navigate to the Site Manager page.
2. On the Site Manager page, click Promotions.
3. On the Promotions Manager page, click Cross Promotions.

► To edit or delete an existing promotion
1. Use your browser to navigate to the Site Manager page.
2. On the Site Manager page, click Promotions.
3. On the Promotions Manager page, click Cross Promotions.
4. On the Cross Promotions page, click the name of the cross promotion you want to edit or delete.
Lab 3: Customizing Product Catalog Pages

After completing this lab, you will be able to:

- Create a customized catalog for your site.
- Add the Cross-sell functionality to your site using the Promotions Wizard.
1. What is the role of Global.asa in a Commerce Server site?

Each Commerce Server site contains a configuration file named global.asa that initializes the site by creating objects such as FileDocument and QueryMap, which are used by a Commerce Server site.

2. What is the advantage of adding search capabilities to your site?

Adding search capabilities will make it easy for shoppers to look for specific information in the catalog.

3. Why is it a good idea to display related products on the site?

Displaying related products on a site encourages shoppers to look at other products available on a site and consider purchasing them.
Instructor Notes Module 4: Managing a Shopping Cart

Introduction

This module provides students with the knowledge necessary to maintain a virtual shopping cart and implement Price promotions and Upsell features on a site.

After completing this module, students will be able to:

- Explain the need for identifying shoppers in a site.
- Generate a shopper ID and assign it to a shopping session.
- Add, remove, and update items in a shopping cart.
- Use Promotions Wizard to add price discounts and special offers on products.
- Implement the Upsell functionality.

Materials and Preparation

This section provides you with the materials and preparation needed to teach this module.

Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file P04_1588.ppt
- Module 4, “Managing a Shopping Cart”
- Lab 4, “Implementing Upsell”

Preparation

To prepare for this module, you should:

- View the relevant .asp files of the Five Lakes Publishing sample site.
- View the relevant Web pages of the Five Lakes Publishing sample site.
- Complete the practice and lab in Module 4.
- Read the relevant sections of the product documentation.
Module Strategy

Use the following strategy to present this module:

- **Managing a Shopping Session**
  Explain to the students that shopping cart information for all shoppers is stored in one table and information for a specific shopper is retrieved using the shopper ID. Describe the code in i_shop.asp that creates a new shopper ID.

- **Adding Items to a Shopping Cart**
  Explain the role of the *OrderForm* object and the *DBStorage* object in a shopping process. Mention that once a shopper selects an item from the product catalog and adds it to the shopping cart, the information is stored in the database and retrieved every time the shopper views the shopping cart page. Describe the code to add an item to a shopping cart.

- **Displaying, Updating, and Removing Items**
  Explain that a shopper can change the cart contents any number of times until the actual purchase. Every change in the cart will result in a change in the *<Sitename>_basket* table in the site database. Describe the code for updating item quantity, removing an item from the shopping cart, and displaying the contents of a shopping cart.

- **Implementing Price Promotions**
  Discuss how price promotions can be applied on all products or products that meet specific criteria. Explain the two ways of adding price promotions to a site using the Promotions Wizard.

- **Implementing Upsell**
  Compare the Cross-sell functionality with Upsell. Mention that in the Five Lakes Publishing sample site, a product can have multiple cross-sell related products but only one Upsell related product. Explain the elements to be included in a site for implementing Upsell. Describe the code in basket.asp, product_alt.asp and xt_orderform_edititem.asp of the Five Lakes Publishing sample site.
Module 4: Managing a Shopping Cart
Overview

Slide Objective
To provide an overview of the module topics and objectives.

Lead-in
In this module, you will learn how to manage a shopping cart and implement Price promotions and Upsell promotions on your site.

- Managing a Shopping Session
- Adding Items to a Shopping Cart
- Displaying, Updating, and Removing Items
- Implementing Price Promotions
- Implementing Upsell
- Lab 4: Implementing Upsell
- Review

Objectives
After completing this module, you will be able to:

- Explain the need for identifying shoppers in a site.
- Generate a shopper ID and assign it to a shopping session.
- Add, remove, and update items in a shopping cart.
- Use Promotions Wizard to add price discounts and special offers on products.
- Implement the Upsell functionality.
Managing a Shopping Session

Slide Objective
To provide an overview of the topics in this section.

Lead-in
Commerce Server associates items on a page with a shopper.

- Identifying Shoppers Through a Shopper ID
- Generating and Storing Shopper IDs
Identifying Shoppers Through a Shopper ID

Slide Objective
To explain the need for identifying shoppers with a shopper ID.

Lead-in
During a shopping session, a shopper is redirected from one page to another.

- Need for identifying shoppers
- What is a shopper ID?

During a visit to a Commerce Server site, a shopper navigates through a number of pages, browsing the product catalog, adding, updating, and removing items from a shopping cart. The contents of a shopping cart are saved and retrieved from the site database. As a shopper moves from one page to another, the site must be able to identify the shopper in order to connect the shopper with the shopping cart data.

A Commerce Server site keeps track of shoppers by assigning each shopper a unique shopper ID. A shopper ID is a random, unique, 32-character string generated by Commerce Server to keep track of a shopper's order. The site uses this ID to maintain orders in the site database.
Generating and Storing Shopper IDs

A Commerce Server site can save a shopper ID from page to page either in a cookie or in the URL.

Storing a shopper ID in a cookie
A cookie is a small file (approximately 1 KB) that contains information about a shopper and is stored on the shopper’s computer. When a shopper connects to a site for the first time, the shopper ID and site address is stored as a cookie on the hard disk of the shopper’s computer. When the shopper moves to a new page on the site, a script in the new page retrieves the shopper ID from the cookie.

Storing a shopper ID in the URL
Another method of saving shopper IDs across pages is by storing it in the URL, so that when a shopper moves from one page to another, the script in the page can extract the shopper ID from the URL. However, to save a shopper ID in this fashion, you must initialize an Application MSCSSIDUrlKey variable to identify the key that will appear in the name/value pair that is stored in the URL.

For example, in the Five Lakes Publishing sample site, the following code in global.asa initializes the Application MSCSSIDUrlKey variable to mscssid.

Application (”MSCSSIDUrlKey”) = “mscssid”
A call to the Page object’s SURL or URL method, to move to a new page, will generate the URL as:

http://host- name/site/newpage.asp?mscsid=TDPLPEI0ULSH2J2400G09V85SGAHDJQB

Creating shopper IDs using StandardSManger

Commerce Server uses the Page object GetShopperID and PutShopperID methods to retrieve and store shopper IDs. These methods store and retrieve shopper IDs either from a cookie or the URL depending on how the StandardSManger object has been initialized in global.asa. StandardSManger is a Commerce Server object that supports methods to create, delete, and retrieve shopper IDs.

You can initialize a StandardSManger object to the cookie mode, as follows:

```vbscript
set mscsShopperManager = Server.CreateObject(“Commerce.StandardSManager”) call mscsShopperManager.InitManager(sitename,”cookie”)```

The following table describes initialization modes of the StandardSManger object.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cookie</td>
<td>Stores and retrieves a shopper ID from a cookie.</td>
</tr>
<tr>
<td>url</td>
<td>Stores and retrieves a shopper ID from the URL.</td>
</tr>
<tr>
<td>Cookieurl</td>
<td>Stores a shopper ID in a cookie and as part of the URL. Retrives a shopper ID from the cookie. If the cookie is not found, the shopper ID is retrieved from the URL.</td>
</tr>
<tr>
<td>Urlcookie</td>
<td>Stores a shopper ID in a cookie and as part of the URL. If the shopper ID is not passed in the URL, it is retrieved from a cookie.</td>
</tr>
</tbody>
</table>

The design of your site will dictate when a shopper ID is generated. You can generate a shopper ID as soon as a shopper enters your site or when a shopper adds an item to the shopping cart. For example, in the Five Lakes Publishing sample site, a shopper ID is generated immediately after a shopper connects to the site.

The following code in i_shop.asp, which is included in most of the pages, creates a shopper ID if it does not exist:

```vbscript
mscsShopperID=mscsPage.GetShopperID
if IsNull(mscsShopperID) then
    mscsShopperID=mscsShopperManager.CreateShopperID()
    mscsPage.PutShopperID(mscsShopperID)
end if```
◆ Adding Items to a Shopping Cart

**Slide Objective**
To provide an overview of the topics in this section.

**Lead-in**
Commerce Server stores shopping cart data in memory and in the site database.

- Using the OrderForm Object
- Understanding the DBStorage Object
- Adding an Item to the Shopping Cart
A shopper who visits your site selects items to purchase in a way that is similar to a shopper walking into a physical store, collecting items in a shopping cart. In a Commerce Server site, a shopper moves from one page to another collecting items in a virtual shopping cart, called an order form.

An order form consists of a set of Dictionary (a list of name/value pairs) objects and SimpleList (an array of variants) objects that provide for the in-memory storage of shopper and purchase information such as shopper ID, items selected for purchase, shipping and billing information, payment information, and various data generated by pipeline components.

Creating an OrderForm object

An **OrderForm** object is created, as follows:

```
set mscsOrderForm = Server.CreateObject(“Commerce.OrderForm”)
```

The following table lists the methods supported by an **OrderForm** object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddItem</td>
<td>Adds an item to the order form.</td>
</tr>
<tr>
<td>ClearOrderForm</td>
<td>Clears the entire <strong>OrderForm</strong> object.</td>
</tr>
<tr>
<td>ClearItems</td>
<td>Clears the items collection from the order form.</td>
</tr>
</tbody>
</table>

Commerce Server sites store the contents of an order form in an order form table in the site database. This table, called `<Sitename>_basket`, contains information of all the shopping carts currently being used by shoppers. Permanent records of completed orders are stored in `<Sitename>_receipt` and `<Sitename>_receipt_item` table.

Not all of the values in the **OrderForm** object are stored in the site database. Those values whose name begins with an underscore (“_”) appear only in the **OrderForm** object and are not saved in the order form table for orders in
progress. These values are saved instead to the site’s receipt table when the purchase is finalized.

Managing an OrderForm object

An OrderForm object is created, loaded, modified, and stored in the site database on a page-by-page basis. As a shopper adds or removes items from the order form, the revised order form data is saved in the database. When the shopper moves to a new page using the shopper ID, the order form values are retrieved from the database and a new OrderForm object is created by using these values.
Understanding the DBStorage Object

Information contained in the OrderForm object is loaded and saved in the site database by means of a DBStorage object.

A DBStorage object is a Commerce Server object that is used to create a data object (Dictionary/SimpleList) that serves as an interface between the site database and the OrderForm object. The DBStorage object provides methods for reading and writing order form values from the site database.

A DBStorage object can be created as follows:

```vbscript
set mscsOrderFormStorage = Server.CreateObject(“Commerce.DBStorage”)
```

A DBStorage object must be initialized to establish a link to the site database and to identify the type of object to be read or written. When you initialize a DBStorage object, you specify:

- The datasource that connects to the site database.
- The table that will contain the order form information.
- The column to be used as the primary key.
- The object in which data will be stored or retrieved by the DBStorage object.
- The column in which the order form values are stored in an encoded format.
- The column to store the date when the data is changed.
A DBStorage object can be initialized, as follows:

```java
call mscsOrderFormStorage.InitStorage(mscsDataSource, ¬
<Sitename>_basket, "shopper_ID", "Commerce.OrderForm", ¬
"marshalled_order", "date_changed")
```

Most of the order form data is stored in the database as a single binary object, usually in the **marshalled_order** field. Queries cannot be run directly on information stored in a binary object. To execute queries on order form data, you can add columns to the table that contains order form data. If a column name exactly matches the name of an entry on the order form, the DBStorage object automatically saves the value for that entry into the column of the same name. For example, if your table contains a column named `ship_to_country`, the DBStorage object saves the value in the `ship_to_country` entry of the order form into that column.
Adding an Item to the Shopping Cart

When a shopper adds an item to a shopping cart, the Commerce Server site executes the following steps to add the item to the site database.

1. A **DBStorage** object is created and initialized to access the site database.
2. The **DBStorage.GetData** method is invoked to retrieve the shopping cart data for the current shopper. Assuming the shopping cart exists, it is returned as an **OrderForm** object by the **GetData** method. If it does not exist, the site creates a new **OrderForm** object.
3. The item is added to the order form by using the **AddItem** method of the **OrderForm** object.
4. If the shopping cart does not exist and a new **OrderForm** object is created, the data is inserted in the site database by using the **DBStorage.InsertData** method. If the order form already exists in the site database, the revised data is updated by using the **DBStorage.CommitData** method.
The following ASP code implements the steps to add an item to the 
<Sitename>_basket table that maintains shopping cart data:

REM -- Create and initialize a DBStorage object
set mscsOrderFormStorage = Server.CreateObject("Commerce.DBStorage")
call mscsOrderFormStorage.InitStorage(mscsDataSource, 
    <Sitename>_basket, "shopper_ID", "Commerce.OrderForm", 
    "marshalled_order", "date_changed")

REM -- Create and initialize an OrderForm object
created = 0
set mscsOrderForm = mscsOrderFormStorage.GetData(null, 
    mscsShopperID)
if IsEmpty(mscsOrderForm) then
    set mscsOrderForm = Server.CreateObject("Commerce.OrderForm")
    mscsOrderForm.shopper_ID = mscsShopperID
    created = 1
end if

REM -- Add the item to the shopping cart
call mscsOrderForm.AddItem(product_sku, product_qty, 0)

REM -- Update the site database
if created = 0 then
    call mscsOrderFormStorage.CommitData(NULL, mscsOrderForm)
else
    call mscsOrderFormStorage.InsertData(NULL, mscsOrderForm)
end if

A Commerce Server site created by using the Site Builder Wizard posts data to xt_orderform_additem.asp when a shopper clicks Add to Basket. The script in xt_orderform_additem.asp saves the data in the OrderForm and updates the database.

For the complete code of xt_orderform_additem.asp created in the Five Lakes Publishing sample site, see Appendix A.
Displaying, Updating, and Removing Items

Slide Objective
To provide an overview of the topics in this section.

Lead-in
During the shopping process, a shopper could be...

- Updating Items in a Shopping Cart
- Removing Items from a Shopping Cart
- Displaying a Shopping Cart
A shopper may want to make changes in the shopping cart. For example, in the Five Lakes Publishing sample site, the shopper can modify the quantity of all items in the shopping cart. Once the shopper has made the changes, the site database needs to be updated with the new values.

A Commerce Server site executes the following steps to update the database with details of the modified item.

1. A `DBStorage` object is created and initialized to access the site database.
2. The `DBStorage.GetData` method is invoked to retrieve the shopping cart data for the current shopper into an `OrderForm` object.
3. The modified values entered by the shopper are updated in the order form.
4. The revised order form data is stored in the site database by using the `DBStorage.CommitData` method.
Assuming that a shopper has modified the quantity of an item in the shopping cart, the following ASP code implements the steps to update the site database:

REM - Create and initialize a DBStorage object
set mscsOrderFormStorage = Server.CreateObject("Commerce.DBStorage")
call mscsOrderFormStorage.InitStorage(mscsDataSource, "<Sitename>_basket", "shopper_ID", "Commerce.OrderForm", "marshalled_order", "date_changed")

REM - Create and initialize an OrderForm object
set mscsOrderForm = mscsOrderFormStorage.GetData(null, mscsShopperID)

REM - Retrieve the Items list from the order form
set items = mscsOrderForm.Items

REM - For every item in the shopping cart page, starting with the last item
for index = mscsOrderForm.Items.count - 1 to 0 step -1
    REM - Retrieve an item
    set item = items(index)
    REM - Retrieve the new quantity value from the shopping cart page
    new_quantity = mscsPage.RequestNumber("qty_" & CStr(index), item.quantity, 0, 999)
    if new_quantity = 0 then
        call mscsOrderForm.Items>Delete(index)
    else
        item.quantity = new_quantity
    end if
    REM -- Update the shopping cart table
    call mscsOrderFormStorage.CommitData(null, mscsOrderForm)
next

The Site Builder Wizard generates xt_orderform_editquantities.asp, which contains the script for updating every row in the <Sitename>_basket table whose quantity has been modified by the shopper.

For the complete code of xt_orderform_editquantities.asp created in the Five Lakes Publishing sample site, see Appendix A.
Removing Items from a Shopping Cart

A shopper can remove an item from the shopping cart. This requires the deletion of the item from the OrderForm object.

A Commerce Server site executes the following steps to delete an item from the OrderForm object and update the site database.

1. A DBStorage object is created and initialized to access the site database.
2. The DBStorage.GetData method is invoked to retrieve the shopping cart data for the current shopper into an OrderForm object.
3. The item is deleted from the OrderForm object.
4. The revised order form data is stored in the site database by using the DBStorage.CommitData method.
The following ASP code deletes an item from the **OrderForm** object and updates the site database:

```vbnet
REM - Create and initialize a DBStorage object
set mcsOrderFormStorage = Server.CreateObject("Commerce.DBStorage")
call mcsOrderFormStorage.InitStorage(mcsDataSource, <Sitename>_basket, "shopper_ID", "Commerce.OrderForm", "marshalled_order", "date_changed")

REM - Create and initialize an OrderForm object
set mcsOrderForm = mcsOrderFormStorage.GetData(null, mcsShopperID)

REM - Delete the item in the shopping cart
call mcsOrderForm.Items.Delete(index)

REM -- Update the shopping cart table
call mcsOrderFormStorage.CommitData(null, mcsOrderForm)
```

The Site Builder Wizard generates `xt_orderform_delitem.asp`, which contains the script for deleting a specific item in the `<Sitename>_basket` table. This script is executed when a shopper removes an item from the shopping cart.

For the complete code of `xt_orderform_delitem.asp` created in the Five Lakes Publishing sample site, see Appendix A.
When a shopper adds, updates, or deletes items in the order form, the site redirects the shopper to the shopping cart page to view the new contents.

A Commerce Server site executes the following steps to retrieve items selected for purchase by the current shopper.

1. A **DBStorage** object is created and initialized to access the site database.
2. The **DBStorage.GetData** method is invoked to retrieve the shopping cart data for the current shopper into an **OrderForm** object.
3. The order form data is displayed on the shopping cart page.

The following ASP code retrieves order form values from the `<Sitename>_basket` table:

```asp
REM - Create and initialize a DBStorage object
set mscsOrderFormStorage = Server.CreateObject("Commerce.DBStorage")
call mscsOrderFormStorage.InitStorage(mscsDataSource, <Sitename>_basket, "shopper_ID", "Commerce.OrderForm", "marshalled_order", "date_changed")

REM - Create and initialize an OrderForm object
set mscsOrderForm = mscsOrderFormStorage.GetData(null, mscsShopperID)
```

The basket.asp file generated by the Site Builder Wizard contains the code to display items on the shopping cart page.

For the complete code of basket.asp created in the Five Lakes Publishing sample site, see Appendix A.
Implementing Price Promotions

Slide Objective
To provide an overview of the topics in this section.

Lead-in
The Site Manager page of your site enables you to offer price promotions on products.

- Understanding Price Promotions
- Using Promotions Wizard
Understanding Price Promotions

Slide Objective
To define a price promotion and explain its types.

Lead-in
A Commerce Server site may want to offer price promotions on products.

What is price promotion?
A price promotion applies price discounts and special offers based on the purchase of products in an order. For example, in the Five Lakes Publishing sample site, a shopper gets a discount of $5 on Programming Visual C++ if the shopper purchases Mastering Windows NT Server 4.

Types of price promotions
Commerce Server enables you to implement different types of price promotions.

The following table describes a few of the many possible types of price promotions.

<table>
<thead>
<tr>
<th>Price Promotion</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Line-Item Discount</td>
<td>Buy one product and get another product at a discount.</td>
</tr>
<tr>
<td>Cross-Department Discount</td>
<td>Buy one product and get a product from a specified department at a discount.</td>
</tr>
<tr>
<td>Two for the Price of One</td>
<td>Buy 2 of a specified product for the price of one.</td>
</tr>
<tr>
<td>Targeted Quantity Discount</td>
<td>Buy a specified number of units of a specified product and get a discount on the order.</td>
</tr>
<tr>
<td>Total Order Percentage Discount</td>
<td>Place an order of more than a specified total price and receive a specified percentage discount on the total order.</td>
</tr>
</tbody>
</table>
The Promotions Wizard of Commerce Server enables you to include price promotions in your site.

► To add a price promotion
1. Go to the Site Manager page by using your Web browser.
2. On the Site Manager page, click Promotions.
3. On the Promotion Manager page, click Price Promotions.

The Price Promotions page enables you to include price promotions based on:
- Quantity of a specific product
- Amount of purchase, quantity of product(s) or products that match a specific criteria

► To enable price promotions based on quantity of a specific product
1. On the Price Promotions page, select one of the following categories from the Add Special Promotion drop-down list box and click Add.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy x get y at z% off</td>
<td>Applies a percentage discount based on the quantity of a specific product.</td>
</tr>
<tr>
<td>Buy x get y at $z off</td>
<td>Applies a cash discount based on the quantity of a specific product.</td>
</tr>
<tr>
<td>Buy 2 x for the price of 1</td>
<td>Applies a discount of 1 unit on the purchase of 2 units of a specific product.</td>
</tr>
</tbody>
</table>
2. Enter values for the following fields that appear on the New Price Promotions page.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Promo Name</td>
<td>A name to identify the price promotion.</td>
</tr>
<tr>
<td>Status</td>
<td>(On/Off) A field that indicates whether the price promotion is enabled.</td>
</tr>
<tr>
<td>Rank</td>
<td>A priority value between 10 and 100 where 10 is the highest priority value.</td>
</tr>
<tr>
<td>Start date</td>
<td>A date on which promotion is to take effect.</td>
</tr>
<tr>
<td>End date</td>
<td>A date on which the promotion is to end.</td>
</tr>
<tr>
<td>Buy</td>
<td>A field that indicates the product and the quantity of the product for which the promotion is applied.</td>
</tr>
<tr>
<td>Get</td>
<td>A field that indicates the product and the quantity of the product on which the promotion is applied.</td>
</tr>
<tr>
<td>At</td>
<td>A numeric value representing cash/percentage discount to be applied.</td>
</tr>
</tbody>
</table>

3. Click **Add Price Promotion** to add the promotion to the list of price promotions.

**To enable price promotions based on amount of purchase, quantity of product(s) or products that match a specific criteria**

1. On the Price Promotions page, click **Add New Price Promotion**.

2. Enter values for the following fields that appear on the New Price Promotions page.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Promo Name</td>
<td>A name to identify the price promotion.</td>
</tr>
<tr>
<td>Description</td>
<td>A description of the price promotion.</td>
</tr>
<tr>
<td>Status</td>
<td>(On/Off) A field that indicates whether the price promotion is enabled.</td>
</tr>
<tr>
<td>Rank</td>
<td>A priority value between 10 and 100 where 10 is the highest priority value.</td>
</tr>
<tr>
<td>Start date</td>
<td>A date on which promotion is to take effect.</td>
</tr>
<tr>
<td>End date</td>
<td>A date on which the promotion is to end.</td>
</tr>
<tr>
<td>Buy</td>
<td>A field that indicates the quantity or value of purchase of specific product(s) for which the promotion will be available. It includes a pair of option buttons that indicate whether the promotion is based on a specific product(s) or all products.</td>
</tr>
<tr>
<td>Get</td>
<td>A field that indicates the quantity of specific product(s) offered by the promotion. It includes a drop-down list box that determines whether the promotion can apply to the product purchased or only to additional products and a pair of option buttons to indicate whether the promotion is applicable on specific product(s) or all products.</td>
</tr>
<tr>
<td>At</td>
<td>A field that indicates the discount in percentage or money amount.</td>
</tr>
</tbody>
</table>
3. Click **Add Price Promotion** to add the promotion to the list of price promotions.
In this practice, you will use the Promotions Wizard to add price promotions.

Access the Promotions Wizard

1. Start Microsoft® Internet Explorer and type the URL http://localhost/FiveLakes/manager in the address bar of Internet Explorer.
2. On the Site Manager page, in the Merchandising section, click Promotions.
3. On the Promotion Manager page, click Price Promotions.

Add price promotions

1. On the Price Promotions page, click buy x get y at $z off in the Add Special Promotion drop-down list box and click Add.
2. On the New Price Promotion page, type the Promo name as Promotion for Computer Section.
3. Select ON from the Status drop-down list box and the select 10 from the Rank drop-down list box.
4. Type two different dates in the Start date and the End date fields in the following format: mm/dd/yyyy.
5. In the **Buy** field:
   a. Click **Mastering Windows NT Server 4** in the drop-down list box within the Buy field.
   b. Type the quantity as **1** in the text box within the Buy field.

6. In the **Get** field:
   a. Select **Programming Visual C++** from the list box within the Get field.
   b. Type the quantity as **1** in the text box within the Get field.
   c. Click **$5 OFF** in the **at** drop-down list box.

7. Click **Add Price Promotion**.

**View the price promotion on the shopping cart page**

1. On the Site Manager page, in the **System** section, click **Shop Site**.
2. On your site Home page, click **Computers**.
3. On the Department: Computers page, click **Mastering Windows NT Server 4**.
4. On the Product: Mastering Windows NT Server 4 page, click **Add to Basket**.
5. Return to the home page.
6. Repeat steps 2 to 4 to add **Programming Visual C++** to the shopping cart.
7. Notice the discount value, on the shopping cart page.
◆ Implementing Upsell

**Slide Objective**
To provide an overview of the topics in this section.

**Lead-in**
A shopper can replace a product in the shopping cart with an Upsell product.

- Understanding Upsell
- Creating the Upsell Table
- Retrieving Upsell Information
- Adding an Upsell Item
An Upsell promotion encourages shoppers to buy a more expensive item instead of the one in the shopping cart.

If an item in the shopping cart has an Upsell item with it, the script in basket.asp displays an Upsell text prompt on the shopping cart page. The Upsell text prompt is a link that enables the shopper to view the Upsell item information. If the shopper decides to buy, the Upsell item replaces the item in the shopping cart. For example, in the Five Lakes Publishing sample site, if a shopper orders Mastering Windows NT Server 4, the shopping cart page suggests Windows NT 4.0 MCSE Study Guide instead.

In order to implement an Upsell promotion in your site, you need to include the following elements.

- An Upsell promotion table in the site database that specifies which product is to be associated with a given product.
- A query description in global.asa to retrieve data from the Upsell promotion table.
- A script in basket.asp that executes a query to retrieve the Upsell item, if any.
- A variation of the Product page to display the Upsell item, if any.
- A .asp file that executes a script to replace the existing item with the promoted item.
Creating the Upsell Table

The following table lists the attributes of the <Sitename>_promo_upsell table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sku</td>
<td>Product-ID of the product for which an Upsell item exists.</td>
</tr>
<tr>
<td>Upsell_sku</td>
<td>Product-ID of the item that will be promoted.</td>
</tr>
<tr>
<td>Description</td>
<td>Upsell text prompt that will appear as a link on the shopping cart page</td>
</tr>
</tbody>
</table>
Once a shopper has selected an item and added it to the shopping cart:

1. The Upsell item for the selected item must be retrieved from the site database.
2. An Upsell text prompt must be displayed as a link on the shopping cart page, which will enable the shopper to view the Upsell item information.

You can add the following query to global.asa to retrieve data from the Upsell promotion table.
The following script in basket.asp executes the related_products_upsell query for a shopping cart item for which Upsell items exist. The script also displays the Upsell text prompt that serves as a link to a page that displays Upsell item information.

```<% REM -- Execute the upsell query to retrieve the upsell item for an item in the shopping cart cmdTemp.CommandText=Replace(MSCSQueryMap. related_products_upsell.SQLCommand,"1",lineitem.sku) set rsUpsell = Server.CreateObject("ADODB.Recordset") rsUpsell.Open cmdTemp, , adOpenKeyset, adLockReadOnly If rsUpsell.recordcount > 0 then REM - For every item in the shopping cart table do while not rsUpsell.EOF     REM - Check if the Upsell item already exists in the Shopping cart set checkitem = mscsOrderForm.items inbasket = 0 for each row_checkitem in checkitem if row_checkitem.sku = rsUpsell("upsell_sku").Value then inbasket = 1 end if next REM - Display Upsell text prompt and redirect shopper to product_alt.asp if Cbool(inbasket = 0) then %> <a HREF="%mscsPage.URL("product_alt.asp","sku", rsUpsell("upsell_sku").Value,"index", ilineitem,"quantity", orderFormItems.quantity,"dept_id", orderFormItems.[_product_dept_id])"><% = rsUpsell("description")%> </a>%end if REM - Retrieve the next item from the shopping cart table rsUpsell.MoveNext loop end if %>
```

For the complete code of basket.asp created in the Five Lakes Publishing sample site, see Appendix A.
Adding an Upsell Item

When a shopper clicks the Upsell text prompt that appears as a link on the shopping cart page, the shopper is redirected to a page that displays the Upsell item information.

In the Five Lakes Publishing sample site, the shopper is redirected to product_alt.asp file that retrieves the Upsell item information from the site database information and displays it on a new page. If the shopper decides to buy, the Upsell item replaces the item on the shopping cart page. In other words, the item already present in the shopping cart is deleted from the order form and the Upsell item is added to the order form. The product_alt.asp file in the Five Lakes Publishing sample site posts data to another .asp file (xt_orderform_edititem) that deletes the shopping cart item and inserts the Upsell item in the site database.

**Note** The product_alt.asp and xt_orderform_edititem.asp files in the Five Lakes Publishing sample site are not wizard-generated files. These files have been created to provide the Upsell functionality in the site.

**To retrieve and display Upsell item information**

1. Define a query to retrieve the Upsell item information from the `<Sitename>_product table.
2. Execute the query when the shopper clicks the Upsell text prompt on the shopping cart page.
3. Redirect the shopper to the page that displays the Upsell item information.

**To update the site database with the Upsell item**

1. Delete the item the shopper wants to replace from the order form.
2. Add the Upsell item to the order form.
For the complete code of product_alt.asp and xt_orderform_edititem.asp created in the Five Lakes Publishing sample site, see Appendix A.
Lab 4: Implementing Upsell

After completing this lab, you will be able to:

- Modify the shopping cart page to include the Upsell functionality.
- Update the site database with the Upsell item.
Review

Slide Objective
To reinforce the module objectives by reviewing key points.

Lead-in
The review questions cover some of the key concepts taught in the module.

- Managing a Shopping Session
- Adding Items to a Shopping Cart
- Displaying, Updating, and Removing Items
- Implementing Price Promotions
- Implementing Upsell
- Lab 4: Implementing Upsell

1. What is the role of an OrderForm object and a DBStorage object in the shopping process?
   An OrderForm object contains shopping cart data and a DBStorage object reads and writes order form values in the site database.

2. Explain the need for creating a shopper ID.
   A shopper ID is used by Commerce Server to associate the shopping cart items and the order with a specific shopper.

3. What change do you need to make in global.asa to store shopper IDs in the URL?
   The StandardSManager object in global.asa must be initialized to the url mode to store shopper IDs in the URL.

4. How is Upsell functionality different from cross-sell?
   Upsell functionality encourages a shopper to buy a more expensive product after the shopper has selected an item for purchase whereas cross-sell encourages a shopper to see related products while browsing the product catalog pages.
Introduction

This module provides students with the knowledge necessary to understand how the Order Processing Pipeline (OPP) works and executes.

After completing this module, the student will be able to:
- Explain what an OPP is.
- Explain how an OPP works.
- List the types of OPP.
- Set up error handling to process errors that occur when the OPP is running.
- Describe the 14 stages of the Plan pipeline.

Materials and Preparation

This section provides you with the materials and preparation needed to teach this module.

Materials
To teach this module, you need the following materials:
- Microsoft® PowerPoint® file P05_1588.ppt
- Module 5, “Processing Orders”

Preparation
To prepare for this module, you should:
- Read all the materials for this module.
- Review the relevant sections of the product documentation.
- Practice running the Pipeline Editor in expert mode and standard mode.
Module Strategy

Use the following strategy to present this module:

- Understanding Order Processing
  Explain how Commerce Server sites process orders. Mention that this module covers only the business-to-consumer Order Processing Pipeline. Give an overview of the five types of OPPs supported by Commerce Server. Demonstrate the Pipeline Editor in expert mode and standard mode. Demonstrate the different ways in which the pipeline configuration can be viewed.

- Running the Order Processing Pipeline
  Describe the code in i_util.asp to explain how the steps to run an OPP are implemented. Explain how errors are handled in Commerce Server.

- Understanding the Plan Pipeline
  Explain the 14 stages of the Plan pipeline. Explain what the components in each of the stages do. Do not talk about the components in the Shipping, Handling, Tax, and Order Total stages at this point, as these are covered in detail in Module 6: Checking Out.
Module 5: Processing Orders
Overview

**Slide Objective**
To provide an overview of the module topics and objectives.

**Lead-in**
In this module, you will learn about Order Processing and an OPP, the steps required to run an OPP, and the Plan pipeline.

- Understanding Order Processing
- Running the Order Processing Pipeline (OPP)
- Understanding the Plan Pipeline
- Review

Objectives

After completing this module, you will be able to:

- Explain what an Order Processing Pipeline (OPP) is.
- Explain how an OPP works.
- List the types of OPP.
- Set up error handling to process errors that occur when the OPP is running.
- Describe the 14 stages of the Plan pipeline.
Understanding Order Processing

Commerce Server sites process orders by placing the information about the order in an OrderForm object and then executing a series of software components that perform operations on the order form. These operations include computing tax or shipping costs for an order and writing an order to receipt storage. The software components that perform these operations are Component Object Model (COM) objects designed to be executed within a Commerce Server pipeline.

Understanding pipelines

A pipeline defines and links a series of stages, each containing pipeline components, and runs them in sequence. You can customize your Commerce Server application by adding, removing, and configuring the components in each stage.

Commerce Server has two pipeline models:

- **Order Processing Pipeline (OPP)** Processes sales order by performing checks on the order form information and verifying payment.
- **Commerce Interchange Pipeline** Enables the secure exchange of business data objects, such as purchase orders and shipping notices, between business trading partners.
How Does OPP Work?

The following are the tasks performed by the OPP.

- The OPP divides the processing of orders into stages and determines the sequence in which work is performed. Stages describe a category of work and each stage in a pipeline consists of zero or more components that run in sequence.

- The OPP coordinates the work of components by ensuring that every component in an OPP is reading from and writing to the same `OrderForm` object.

- After each component in the OPP has performed its work on the `OrderForm`, the OPP passes the `OrderForm` to the next component in the pipeline.
What are the Types of OPP?

Commerce Server has five basic types of OPPs.

**Business-to-consumer pipelines**

The business-to-consumer OPPs are used in e-commerce sites designed for retail shopping. The following table lists and describes the business-to-consumer pipelines.

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Product</strong></td>
<td>Runs OPP components that compute price and discount information on individual products.</td>
</tr>
<tr>
<td><strong>Plan</strong></td>
<td>Runs OPP components that display order total, including promotional discounts, taxes, shipping and handling charges to the shopper.</td>
</tr>
<tr>
<td><strong>Purchase</strong></td>
<td>Runs OPP components that validate the shopper’s payment, perform the actual purchase transaction, and write an order to database storage.</td>
</tr>
</tbody>
</table>

**Note** The site generated by the Site Builder Wizard uses only the Plan and Purchase pipelines to process an order. The Plan pipeline contains components that compute tax, shipping, and handling charges in addition to the components of the Product pipeline.
Business-to-business pipelines

The business-to-business OPPs are used in e-commerce sites designed for creation and exchange of purchase orders between business trading partners. The following table lists and describes the business-to-business pipelines.

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corporate Purchasing Plan</td>
<td>Runs OPP components that compute the order total, including promotional discounts, taxes, shipping and handling charges. This pipeline is analogous to the Plan pipeline used in business-to-consumer sites.</td>
</tr>
<tr>
<td>Corporate Purchasing Submit</td>
<td>Runs OPP components that validate the purchase order requisition, transfer the purchase order to the vendor, and write the order to database storage. This pipeline is analogous to the Purchase pipeline used in business-to-consumer sites.</td>
</tr>
</tbody>
</table>
How to View Pipeline Configuration Files?

The pipeline components along with their properties are saved in a pipeline configuration (.pcf) file. This file is created in the /Config folder of a custom site created by the Site Builder Wizard. For example, Plan.pcf contains the Plan pipeline configuration.

You can create, modify, and view the configuration files of sites by using the Pipeline Editor. Alternatively, you can configure the pipelines of a specific site by using the Site Manager page.

Using the Pipeline Editor

The Pipeline Editor displays a tree, outlining the stages implemented by the default OPP, and the components that can be used in each stage of processing. Stages and components are executed by the OPP in the order in which they are included in the tree.

The Pipeline Editor can be run either in standard mode or expert mode. To run the Pipeline Editor in standard mode, use one of the following methods:

- On the Start menu, point to Programs, Microsoft Site Server, Commerce, and then click Pipeline Editor.
- Invoke PipeEditor.exe from the command prompt.

To run the Pipeline Editor in expert mode, invoke PipeEditor.exe from the command prompt as follows:

PipeEditor.exe /e
Running the Pipeline Editor in **expert mode** provides you with the additional functionality to:

- Create a custom pipeline without using a pre-configured template file.
- Insert, move, and delete pipeline stages.
- Cut, copy, and paste a stage including its components.
- View the required components that are not displayed in the standard mode.

**Using the Site Manager page**

By using the Site Manager page, you can configure the pipelines of a specific site to:

- Insert, edit, or delete components in a stage.
- Change the order in which the components will be executed within a stage.
◆ Running the Order Processing Pipeline (OPP)

Slide Objective
To provide an overview of the topics in this section.

Lead-in
In this section, you will learn about the steps to be performed in order to run an OPP and handle errors.

- Process Overview
- Error Handling
The **OrderForm** object and its corresponding table in the database maintain information about an order throughout a shopping session. Whenever computations need to be performed on the order, the **OrderForm** object is passed to the OPP. The OPP then performs appropriate verifications and computations, and stores updated values into the **OrderForm**.

The following procedure explains the steps that are performed in order to run an OPP. The i_util.asp file contains the script to run an OPP and is executed whenever the order form needs to be processed.

**To run an OPP**

1. Create a DBStorage object for order form data, and then initialize the object as follows:

   ```
   Set orderFormStorage = Server.CreateObject("Commerce.DBStorage")
   OrderFormStorage.InitStorage (MSCSSite.DefaultConnectionString, "<Sitename>_basket", "shopper_id", "Commerce.OrderForm", "marshalled_basket", "date_changed")
   ```
2. Read the order form data by using the current shopper ID and store it into a new **OrderForm** object as follows:

```vba
created = 0
On Error Resume Next
Set orderForm = orderFormStorage.GetData(null, ¬
mscsShopperID)
On Error Goto 0
if IsEmpty(orderForm) then
    set orderForm = ¬
        Server.CreateObject("Commerce.OrderForm")
    orderForm.shopper_id = mscsShopperID
    created = 1
end if
```

3. Create and configure a pipe context as shown in the following code. The **PipeContext** object is a Dictionary object containing a group of initialized objects that store information needed by the OPP.

```vba
Set pipeContext = ¬
    Server.CreateObject("Commerce.Dictionary")
Set pipeContext("MessageManager") = MSCSMessageManager
Set pipeContext("DataFunctions") = MSCSDataFunctions
Set pipeContext("QueryMap") = MSCSQeryMap
Set pipeContext("ConnectionString") = ¬
    MSCSSite.ConnectionStringMap
pipeContext("SiteName") = displayName
pipeContext("DefaultConnectionString") = ¬
    MSCSSite.DefaultConnectionString
pipeContext("Language") = "USA"
```

4. Create the pipeline as follows:

```vba
Set pipeline = Server.CreateObject("Commerce.MtsPipeline")
```

5. Load the appropriate pipeline configuration file into the pipeline by using the **LoadPipe** method as follows:

```vba
Call
pipeline.LoadPipe("e:\inetpub\wwwroot\<Sitename>\plan.pcf")
```

6. Run the pipeline by using the **Execute** method, specifying both the **OrderForm** object and the **PipeContext** object, as follows:

```vba
errorLevel = pipeline.Execute(1, orderForm, pipeContext, 0)
```

The syntax of the **Execute** method is:

```
MtsPipeline.Execute (Mode, Object, PipeContext, Reserved)
```

The following table explains the parameters of the **Execute** method.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Specifies the mode in which to execute the stages in the pipeline. This parameter is included for backward compatibility with sites created in Commerce Server 2.0.</td>
</tr>
<tr>
<td>Object</td>
<td>Specifies the data object to be processed by the components in the pipeline.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>PipeContext</td>
<td>Specifies the PipeContext object.</td>
</tr>
<tr>
<td>Reserved</td>
<td>This value is not used and must be set to zero.</td>
</tr>
</tbody>
</table>

7. Save the **OrderForm** back to the order form table in the database, if running the pipeline made any changes to the order form data.

```vbnet
if created then
    Call mscsOrderFormStorage.InsertData(null, ¬
        mscsOrderForm)
else
    Call mscsOrderFormStorage.CommitData(null, ¬
        mscsOrderForm)
end if
```

For the complete code of i_util.asp in the Five Lakes Publishing sample site, see Appendix A.
Error Handling

Slide Objective
To explain how errors are handled in Commerce Server.

Lead-in
Error handling in Commerce Server revolves around the OPP’s interaction with the OrderForm, MtsPipeline or MtsTxPipeline object, and the MessageManager object.

- A MessageManager component is created in global.asa
- AddMessage method is called to add messages to MessageManager
- MessageManager is initialized as Application variable

You pass an initialized OrderForm object to the OPP by calling the pipeline’s Execute method. As the OrderForm is passed through various stages of the OPP, the components associated with each stage read and write values to and from the OrderForm. The collections _Basket_Errors and _Purchase_Errors, which are members of the OrderForm object, store strings that describe error conditions detected by the OPP while processing elements of the OrderForm. The OPP gets these strings from the MessageManager.

The MessageManager
The MessageManager is a site’s central repository of locale-based error messages. Each message stored in the MessageManager consists of a string describing the error condition and a string identifier. The MessageManager uses the AddMessage method to create the association between string identifier and string message.

For example, if the OPP is unable to validate a credit card in the OrderForm, the pipeline retrieves the string associated with the message ID pur_badcc from the MessageManager. The OPP then writes the string to the OrderForm’s _Purchase_Errors collection. When the Execute method that initiated the pipeline returns, the strings in this collection can be used to provide site users with a description of errors that occurred during order processing.

To set up error handling
1. A MessageManager component is created in the global.asa file as follows:

   REM - Create a message manager for use by the pipeline
   Set MSCSMessageManager = Server.CreateObject(“Commerce.MessageManager”);
2. The MessageManager’s **AddMessage** method is called to add a group of messages to the MessageManager as follows:

```vba
    call MSCSMessageManager.AddMessage("pur_out_of_stock", "At least one item is out of stock.")
    call MSCSMessageManager.AddMessage("pur_badsku", "Please note that one or more items were removed from your basket because the product is no longer sold.")
    call MSCSMessageManager.AddMessage("pur_noitems", "An order must have at least one item.")
    call MSCSMessageManager.AddMessage("pur_badpayment", "There was a problem authorizing your credit. Please verify your payment information or use a different card.")
    call MSCSMessageManager.AddMessage("pur_badcc", "The credit-card number you provided is not valid. Please verify your payment information or use a different card.")
```

3. MessageManager is initialized as an Application variable named **MSCSMessageManager**, so that it can be accessed across all pages, as shown in the following code:

```vba
    set Application("MSCSMessageManager") = MSCSMessageManager
```
Understanding the Plan Pipeline

Slide Objective
To provide an overview of the topics covered in this section.

Lead-in
In this section, you will learn about the stages and components of a Plan pipeline.

- Stages in the Plan Pipeline
- Components in the Plan Pipeline Stages
The Plan pipeline consists of 14 stages. These stages are made up of components that verify the integrity of the OrderForm. For example, if the items list in an OrderForm does not contain any item, then the RequiredProdInfo component in the Product Info stage fails.

The following table lists and describes each stage of the Plan pipeline.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Info</td>
<td>Contains components that retrieve product information about the items from the site database.</td>
</tr>
<tr>
<td>Merchant Information</td>
<td>Can contain custom components to retrieve merchant data and write the data to the OrderForm.</td>
</tr>
<tr>
<td>Shopper Information</td>
<td>Adds information about the shopper to the OrderForm.</td>
</tr>
<tr>
<td>Order Initialization</td>
<td>Sets initial order information on the OrderForm and verifies that the OrderForm contains an Order ID.</td>
</tr>
<tr>
<td>Order Check</td>
<td>Verifies that the order can be processed.</td>
</tr>
<tr>
<td>Item Price</td>
<td>Ensures that the _iadjust_regularprice for each item contains the most current price information.</td>
</tr>
<tr>
<td>Item Adjust Price</td>
<td>Ensures that the _iadjust_currentprice for each item contains the current price adjusted for sales or promotions.</td>
</tr>
<tr>
<td>Order Adjust Price</td>
<td>Contains components that set the adjusted price of each item.</td>
</tr>
<tr>
<td>Order Subtotal</td>
<td>Calculates the subtotal for an order.</td>
</tr>
<tr>
<td>Shipping</td>
<td>Calculates the total shipping charge for the order.</td>
</tr>
<tr>
<td>Stage</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Handling</td>
<td>Calculates the total handling charge for the order.</td>
</tr>
<tr>
<td>Tax</td>
<td>Computes the sales tax for each item on the order and the sum of the tax for the entire order.</td>
</tr>
<tr>
<td>Order Total</td>
<td>Sums the subtotal, tax, shipping, and handling values.</td>
</tr>
<tr>
<td>Inventory</td>
<td>Verifies that every item ordered is in stock.</td>
</tr>
</tbody>
</table>
You can configure the Plan pipeline by inserting, deleting, and modifying the stages and their components to suit the needs of your site. The following table lists and describes the components in the various stages of the Plan pipeline used in the Five Lakes Publishing sample site.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Info</td>
<td>QueryProdInfo</td>
<td>Executes a database query, based on the SKUs in the OrderForm’s item list and puts the returned data in the items collection. If data is not found for an SKU, the item is marked for deletion.</td>
</tr>
<tr>
<td>Product Info</td>
<td>RequiredProdInfo</td>
<td>Deletes any item that is marked for deletion.</td>
</tr>
<tr>
<td>Shopper Information</td>
<td>DefaultShopperInfo</td>
<td>Initializes entries in the OrderForm to contain values about the shopper.</td>
</tr>
<tr>
<td>Order Initialization</td>
<td>RequiredOrderInit</td>
<td>Initializes values in the OrderForm prior to processing the order.</td>
</tr>
<tr>
<td>Order Check</td>
<td>RequiredOrderCheck</td>
<td>Ensures that there is at least one item in the OrderForm.</td>
</tr>
<tr>
<td>Item Price</td>
<td>DefaultItemPrice</td>
<td>Initializes _iadjust_regularprice for each item in the items list to store the most current item price.</td>
</tr>
<tr>
<td>Item Price</td>
<td>RequiredItemPrice</td>
<td>Ensures that _iadjust_regularprice is set.</td>
</tr>
<tr>
<td>Stage</td>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Item Adjust Price</td>
<td>SaleAdjust</td>
<td>for each item in the items list. Determines whether an item is on sale and sets the _iadjust_currentprice name/value pair to the current price of the item, adjusted for sales or promotions.</td>
</tr>
<tr>
<td>Item Adjust Price</td>
<td>RequiredItemAdjustPrice</td>
<td>Verifies that the _iadjust_currentprice name/value pair is set.</td>
</tr>
<tr>
<td>Order Adjust Price</td>
<td>DbOrderPromoADO</td>
<td>Queries the database to determine the promotion amount for an item.</td>
</tr>
<tr>
<td>Order Adjust Price</td>
<td>RequiredOrderAdjustPrice</td>
<td>Ensures that the _oadjust_adjustedprice member of every item, containing total cost of an item, is set.</td>
</tr>
<tr>
<td>Order Subtotal</td>
<td>DefaultOrderSubtotal</td>
<td>Calculates the subtotal for an order, and stores the result in order_oadjust_subtotal.</td>
</tr>
<tr>
<td>Order Subtotal</td>
<td>RequiredOrderSubtotal</td>
<td>Ensures that order_oadjust_subtotal is not NULL.</td>
</tr>
</tbody>
</table>

For more information about the other stages of the Plan pipeline, see “Module 6: Checking Out.”
1. How can you run the Pipeline Editor in expert mode?
   To run the Pipeline Editor in expert mode, invoke PipeEditor.exe from the command prompt as follows:
   
   PipeEditor.exe /e

2. What are the advantages of running the Pipeline Editor in expert mode?
   Running the Pipeline Editor in expert allows you to create a custom pipeline without using a pre-configured template file. You can insert, move, and delete pipeline stages including their components as well as view the required components that are not displayed in the standard mode.

3. Which stage of the Plan pipeline calculates the subtotal, tax, shipping, and handling values for an order?
   The Order Total stage calculates the subtotal, tax, shipping, and handling values for an order.

4. Which two collections store the error strings describing the error conditions detected by the OPP while processing elements of the OrderForm?
   The collections _Basket_Errors and _Purchase_Errors store the error strings describing the error conditions detected by the OPP.
Instructor Notes Module 6: Checking Out

Introduction

Presentation: 40 Minutes
Lab: 20 Minutes

This module provides students with the knowledge necessary to add a Scriptor component and customize the shipping page to allow shoppers from all countries to shop at the site.

After completing this module, students will be able to:

- Explain the components in the Plan pipeline that compute tax, shipping, and handling charges, and the total cost of an order.
- Customize the shipping page to allow shoppers from all countries to shop at the site.
- Add a Scriptor component to compute tax.

Materials and Preparation

This section provides you with the materials and preparation needed to teach this module.

Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file P06_1588 .ppt
- Module 6, “Checking Out”
- Lab 6, “Adding a Scriptor Component”

Preparation

To prepare for this module, you should:

- Complete the practice and lab in the module.
- Read the relevant sections of the product documentation.
Module Strategy

Use the following strategy to present this module:

- Capturing Shopper Information
  Explain how a shopper’s ship-to address information is stored in the `OrderForm` object and processed by the pipeline components to compute the total cost of the order.
  Discuss how the shipping page can be modified to allow shoppers from all countries to shop at the site.
  Describe the code in `shipping.asp` file of the sample site that displays a list of country names in the HTML form.

- Computing Order Value
  Explain the Shipping, Handling, Tax, and Order Total stages of the Plan pipeline.
  Talk about how the components in each stage access or modify the values in the `OrderForm` object.
  Mention the components that are added by the Site Builder Wizard.

- Adding a Scriptor Component
  Define a Scriptor component and explain the need for adding a Scriptor component.
  Demonstrate the steps to add and configure a Scriptor component.
  Describe the code in the Tax Calculator Scriptor component in `plan.pcf` file of the sample site.
Module 6: Checking Out
Overview

**Slide Objective**
To provide an overview of the module topics and objectives.

**Lead-in**
In this module, you will learn about the Plan pipeline stages that compute the order value. You will also learn to add a Scriptor component.

- Capturing Shopper Information
- Computing Order Value
- Adding a Scriptor Component
- Lab 6: Adding a Scriptor Component
- Review

Objectives
After completing this module, you will be able to:

- Explain the components in the Plan pipeline that compute tax, shipping and handling charges, and the total cost of an order.
- Customize the shipping page to allow shoppers from all countries to shop at the site.
- Add a Scriptor component to compute tax.
◆ Capturing Shopper Information

**Slide Objective**
To provide an overview of the topics in this section.

**Lead-in**
A site must accept a shopper's address and payment information to complete the purchase.

- Understanding Checkout
- Practice: Customizing the Shipping Page
Understanding Checkout

When a shopper is ready to complete the purchase of the selected items, a Commerce Server site:

- Accepts the shopper’s ship-to address information.
- Stores the shopper information in the `OrderForm` object.
- Executes the Plan pipeline to validate shopper information and compute the total cost of the order.

Accepting ship-to address information

Once a shopper confirms purchase on the shopping cart page, the shopper is redirected to the shipping page (shipping.asp). Here, the shopper enters the ship-to address information in an HTML form or by using Microsoft® Wallet. Wallet provides shoppers an easy and secure way to store address and payment information.

For more information on Wallet, see “Module 7: Completing the Purchase Process.”

Storing information in the OrderForm object

The script in shipping.asp posts the shopper information to `xt_orderform_prepare.asp`, which validates the shopper’s ship-to address information and stores this information in the `OrderForm` object.

Executing the Plan pipeline

Finally, the Plan pipeline is executed to:

- Validate the shopper’s ship-to address information.
- Compute shipping, handling and tax charges for the order and store it in the `OrderForm` object.
- Compute the total cost of the order and store it in the `OrderForm` object.
Practice: Customizing the Shipping Page

In this exercise, you will modify the shipping page to allow shoppers from other countries to shop at your site.

Caution The activities in this practice are a prerequisite for successful completion of lab 6.

Define a query in global.asa

1. In Microsoft Visual InterDev® 6.0, open the FiveLakes project.
2. Double-click global.asa in the Project Explorer window.
3. In the `InitQueryMap` function, add a query to retrieve a list of country names from the FiveLakes_country table as follows:

   ```
   set MSCSQueryMap.country = ^
   AddQuery("select country from FiveLakes_country")
   ```

4. On the File menu, click Save global.asa, and then click Close.

Display the list of country names on the shipping page

1. Double-click shipping.asp in the Project Explorer window.
2. In shipping.asp, delete the following line of code to allow any country to be accepted:

   ```
   <INPUT TYPE="HIDDEN" NAME="ship_to_country" VALUE="USA">
3. Next, scroll to the code that reads as follows:

\[
\text{VALUE} = \text{"%} \!
\text{mscsPage.HTMLEncode(mscsOrderForm.ship_to_zip)} \text{"} > \!
\&nbsp;
\]

</TD>
</TR>

4. Below this code, add code to create an HTML select element that displays a list of country names from the FiveLakes_country table as follows:

<TR>
<TD ALIGN="RIGHT">
<B> Country: </B>
</TD>

<% sqlText = MSCSQueryMap.country.SQLCommand
cmdTemp.CommandText = sqlText
set rsCountry=Server.CreateObject("ADODB.Recordset")
rsCountry.Open cmdTemp, , adOpenForwardOnly, ¬
adLockReadOnly %>

<TD ALIGN="LEFT">
<select name="Ship_to_country" size="1">
</% While not rsCountry.EOF %>
<% mscsPage.Option(rsCountry("country").value, ¬
mscsOrderForm.ship_to_country) %>
<% rsCountry("country").value %>
</select>
Wend %>

</select VALUE="% ¬
mscsPage.HTMLEncode(mscsOrderForm.ship_to_country) ¬
%"%>
</TD>
</TR>

5. On the File menu, click Save shipping.asp, and then click Close.

Modify the Plan pipeline

1. Start Microsoft Internet Explorer and type the URL http://localhost/FiveLakes/manager in the address bar of Internet Explorer.

2. On the Site Manager page, select plan.pcf from the drop-down list box in the System section and click Edit Pipeline.

4. In the **MSCExecute** function within the **Script** text box, delete the following line of code that allows shoppers from only one country to shop at the site:

   ```
   if orderform.ship_to_country <> "USA" then call ¬
   errors.Add(msg_mgr.GetMessage("val_shipcountrymustbe")); ¬
   result = 2
   ```

5. Click **Update** to save the script changes.

6. Click **Save** to save the plan.pcf file.

**View the country list on the shipping page**

1. Start Microsoft Internet Explorer and type the URL **http://localhost/FiveLakes** in the address bar of Internet Explorer.

2. Shop at the site for at least one product and click **Purchase** on the Shopping Basket page.

3. On the Shipping page, click **Click here if you have problems with the Wallet**.

4. Notice the drop-down list box on the shipping page that displays the list of countries.
Computing Order Value

Slide Objective
To provide an overview of the topics in this section.

Lead-in
Once a shopper has entered the ship-to address information, the site runs the Plan pipeline for...

- Computing Shipping and Handling Charges
- Computing Tax
- Computing Order Total
Computing Shipping and Handling Charges

The Shipping stage
The Shipping stage in the Plan pipeline:

- Computes the shipping cost for the order.
- Stores the computed value in the _shipping_total name/value pair in the OrderForm object.

The Handling stage
- Computes the total handling cost for an order.
- Stores the computed value in the _handling_total name/value pair in the OrderForm object.

Commerce Server provides a number of pipeline components to compute the shipping cost.

The following table lists and describes some of the components of the shipping stage.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultShipping</td>
<td>Sets _shipping_total to zero.</td>
</tr>
<tr>
<td>FixedShipping</td>
<td>Sets _shipping_total to a fixed amount.</td>
</tr>
<tr>
<td>TableShippingADO</td>
<td>Sets _shipping_total to a value returned by a specified query.</td>
</tr>
<tr>
<td>RequiredShipping</td>
<td>Checks the order form to ensure that _shipping_total has been set.</td>
</tr>
</tbody>
</table>

Shipping options
Multiple components can be inserted to allow for different shipping costs depending upon the shipping method used. The Site Builder Wizard provides two shipping methods: Overnight and 2nd Day. The wizard includes two instances of the FixedShipping component in the shipping stage. One instance will set the _shipping_total name/value pair to a cost specified for the Overnight shipping method, and the other instance will set it to a cost specified for the 2nd Day shipping method. If you do not want to charge shipping on an
order, you can include only the DefaultShipping component, which sets _shipping_total to zero.

The Handling stage

The handling stage in the Plan pipeline:

- Computes the total handling cost of an order.
- Stores the computed value in the _handling_total name/value pair in the OrderForm object.

The following table lists and describes some of the components provided by Commerce Server to compute handling charges.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultHandling</td>
<td>Sets the _handling_total name/value pair to zero.</td>
</tr>
<tr>
<td>FixedHandling</td>
<td>Sets the _handling_total name/value pair to a fixed amount.</td>
</tr>
<tr>
<td>TableHandlingADO</td>
<td>Computes the value for the _handling_total name/value pair based on the results of a query.</td>
</tr>
<tr>
<td>RequiredHandling</td>
<td>Checks the order form to ensure that the _handling_total name/value pair has been set.</td>
</tr>
</tbody>
</table>

Handling options

The Site Builder Wizard provides an option to specify the handling charges for an order. The wizard includes a FixedHandling component that applies the specified handling charges to the order. If you do not want to charge handling on an order, you can include only the DefaultHandling component that sets _handling_total to zero.
Computing Tax

The Tax stage

The Tax stage in the Plan pipeline:

- Computes tax for every item in the order.
- Computes the total tax for the order.
- Sets the _tax_total and _tax_included name/value pairs for every item in the items list of the OrderForm object.
- Sets the _tax_total and _tax_included name/value pairs for the entire order in the OrderForm object.

The following table lists and describes some of the components provided by Commerce Server to compute tax.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultTax</td>
<td>Sets the OrderForm’s _tax_total and _tax_included values to zero.</td>
</tr>
<tr>
<td>SimpleCanadaTax</td>
<td>Applies a specified tax for Canada, including GST (Goods and Services tax) and PST ( Provincial Sales tax).</td>
</tr>
<tr>
<td>SimpleJapanTax</td>
<td>Applies a specified tax for the Japanese model.</td>
</tr>
<tr>
<td>SimpleUSTax</td>
<td>Applies a specified tax to any order from a given state.</td>
</tr>
<tr>
<td>RequiredTax</td>
<td>Checks if the OrderForm object’s _tax_total and _tax_included values are set.</td>
</tr>
</tbody>
</table>

Computing tax on individual items

The Site Builder Wizard allows you to specify a tax rate to be applied for the country you have selected on the Commerce Site Builder Wizard: Locale page. Multiple tax components can be inserted in the pipeline, each one applying a tax rate for a specific country. The country name is specified as a parameter in the tax component. For a tax component to execute, the value in the
OrderForm object’s ship_to_country name/value pair must be set to the country specified within the component.

Even though the component name may include the country name, you need to specify the country name as a parameter because the name of the component only indicates the model for computing tax and not the country to which it applies. For example, the SimpleCanadaTax component can be used for another country whose tax model is similar to Canada.

Every component in the tax stage must check the values in the _tax_total and _tax_included name/value pairs before execution. If these values are already set, the component must return without doing anything. If the values are not set, the component must attempt to compute the tax and if it is successful, set the values in the two name/value pairs. If you are writing a custom tax-processing component, you must ensure that your component writes to _tax_total and _tax_included name/value pairs.
Computing Order Total

The Order Total stage

The Order Total stage in the Plan pipeline:

- Sums the subtotal (less any discount), shipping, handling and tax charges.
- Sets the total value of the order in the OrderForm object.

In other words, this stage reads and sums the values in _oadjust_subtotal, _shipping_total, _tax_total and _handling_total name/value pairs and writes the sum in the _total_total name/value pair in the OrderForm object.

The following table lists and describes the components provided by Commerce Server to store the order total in the order form.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultTotal</td>
<td>Writes the total cost of the order in the _total_total name/value pair.</td>
</tr>
<tr>
<td>RequiredTotal</td>
<td>Performs a _VERIFY_WITH check on the OrderForm object.</td>
</tr>
</tbody>
</table>

Once the Plan pipeline is executed completely, the script in payment.asp reads the updated OrderForm values and displays the complete order along with the shipping, handling, and tax charges on the Final Purchase Approval page.
Adding a Scriptor Component

Slide Objective
To provide an overview of the topics in this section.

Lead-in
A custom component can be created to execute task-specific code.

- Understanding the Scriptor Component
- Adding a Scriptor Component to Compute Tax
Understanding the Scriptor Component

A Scriptor component is a custom component that can be inserted in any stage of a pipeline. The code in a Scriptor component can be written by using either the Microsoft Visual Basic® Scripting Edition or Microsoft Jscript® scripting language.

A Scriptor component allows you to:

- Access and modify the OrderForm object.
- Access the pipe context information.
- Execute a pipeline.

The script run by a Scriptor component can be stored internally as part of the .pcf file, or externally in a separate file that is called by the component when it runs.

**To add a Scriptor component**

1. Start Microsoft Internet Explorer and type the URL `http://localhost/Sitename>/manager` in the address bar of Internet Explorer.
2. On the Site Manager page, under the System section, select the .pcf file to which you want to add the Scriptor component, and then click Edit Pipeline.
3. Scroll to the stage to which you want to add the Scriptor component and click Insert component.
4. Select Scriptor from the list of available components.

Once you have added a Scriptor component to a pipeline, you need to configure the component to specify:

- The scripting language to write the code.
- The location of the script that indicates whether the script will be stored internally in the .pcf file or externally in a file.
- The parameters that will be passed to the script.

▶ To configure a Scriptor component
1. Add a Scriptor component to the pipeline.
2. Scroll to the stage to which you added the Scriptor component and click the Edit link that appears under the component.
3. Select a scripting language from the Scripting Engine drop-down list box.
4. Select the Source as Internal or External.
5. If the Source is Internal, type the code in the Script box. If the Source is External, type the path and filename that contains the script, in the filename box.
6. Type the parameters in the Config box, in the form name=value.
7. Click Update to save the values in the .pcf file.
Adding a Scriptor Component to Compute Tax

Slide Objective
To explain the need for creating a tax component and the logic for computing tax.

Lead-in
You can create a single component that computes tax for multiple countries.

The Site Builder Wizard requires a locale value for your site. The wizard uses this locale value to configure your site to compute tax and display time and currency in the proper locale format. For example, if you select English (United States) as the locale for your site, the tax charges for an order will be computed based on the tax rates applicable in United States.

Computing tax based on the shopper’s location
If you want your site to cater to shoppers from other countries, you can create a Scriptor component that performs processing based on a shopper’s country. For example, you can include a Scriptor component in the Tax stage that computes tax based on the country name stored in the OrderForm object’s ship_to_country name/value pair. The tax component must ensure that it sets the _tax_total and _tax_included name/value pairs in the OrderForm object.

Understanding the Scriptor component
When you configure a Scriptor component by using the Site Manager page, select the Scripting engine as VBScript and select the source as Internal. Three functions are inserted in the Script text box: MSCSOpen, MSCSEexecute and MSCSClose. These functions are called the entry points of a Scriptor component.

- The MSCSOpen function
  This function is executed immediately when the Scriptor component is run. It receives as a parameter, the configuration dictionary information such as the script text and the script name.

- The MSCSEexecute function
  This function is executed immediately after the MSCSOpen function and receives the following parameters:
  - The configuration dictionary information such as the script text and the script name.
The OrderForm dictionary.
- The pipe context dictionary.
- The flags that you pass to the pipeline object.

- The MSCSClose function
  This function is executed after the MSCSEexecute function has returned.

You can write your own task-specific code within these functions. The Five Lakes Publishing sample site contains an Internal Scriptor component called Tax Calculator in the Tax stage of the Plan pipeline. The following script in the Tax Calculator component computes the tax for each item in the OrderForm object's items list and for the complete order:

```pascal
function MSCSEexecute(config, orderform, context, flags)
  item_tax_rate = orderform.tax_rate
  set itemlist = orderform.items
  for each item in itemlist
    item.[_tax_total] = (item.list_price*item_tax_rate)/100
    item.[_tax_included] = 0
    orderform.[_tax_total] = item.[_tax_total] +
    orderform.[_tax_total]
  next
  MSCSEexecute = 1  'set function return value to 1 for success
end function
```
Lab 6: Adding a Scriptor Component

Slide Objective
To prepare students for the lab.

Lead-in
In this lab, you will add a Scriptor component in the Tax stage to compute tax.

After completing this lab, you will be able to modify the Plan pipeline to accommodate shoppers from other countries.
1. How is the total tax for an order computed?
   The components in the Tax stage apply a tax rate specified for the shopper’s country to every item in the order and write the total tax to the OrderForm object.

2. What are the charges included in the total order value?
   The total order value includes the order subtotal, tax, shipping, and handling charges computed by the stages in the Plan pipeline.

3. Explain the need to configure a Scriptor component.
   Once you have added a Scriptor component to a pipeline, you need to configure the component to specify the scripting language to write the code, the location of the script and the parameters that will be passed to the script.

4. What can you do with a Scriptor component?
   A Scriptor component allows you to access and modify the OrderForm object, access the pipe context information and execute a pipeline.
Instructor Notes Module 7: Completing the Purchase Process

Introduction

This module provides students with the knowledge necessary to complete the purchase stage in the shopping process, generate simple order numbers, and track an order.

After completing this module, students will be able to:

- Capture payment information in an HTML form or using Microsoft Wallet.
- Generate simple order numbers.
- Track the status of an order.
- Explain how business transactions are secured using HTTPS and Digital Certificates.

Materials and Preparation

This section provides you with the materials and preparation needed to teach this module.

Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file P07_1588.ppt
- Module 7, “Completing the Purchase Process”
- Lab 7, “Tracking Order Status”

Preparation

To prepare for this module, you should:

- Read all the materials for this module.
- Review the relevant sections of the product documentation.
- Complete the practice and lab in this module.
Module Strategy

Use the following strategy to present this module:

- Understanding Purchase
  Recap the shopping stages before a shopper reaches the purchase stage. Explain the advantages of Microsoft Wallet over HTML forms.
  Describe the code in xt_orderform_purchase.asp to show the execution of the Purchase pipeline after shopper information is validated.

- Executing the Purchase OPP
  Explain why the Purchase OPP is run as a transacted pipeline. Explain the importance of the MtsTxPipeline object. Describe the code in the Validate Bill_to, Validate CC_Info and SQLItemADO components in purchase.pcf.

- Tracking an Order
  Discuss different methods of generating simple order numbers that can be remembered easily.

- Securing Business Transactions
  Using the Five Lakes Publishing sample site as an example, explain the need for xt_orderform_purchase.asp to be a secure page since it receives confidential information such as credit card numbers from payment.asp.
  The slide titled Introducing Digital Certificates contains an animation that shows how digital certificate work. To run the animation, click the graphic on the slide.
Module 7: Completing the Purchase Process
Overview

Objectives
After completing this module, you will be able to:

- Capture payment information in an HTML form or by using Microsoft® Wallet.
- Generate simple order numbers.
- Track the status of an order.
- Explain how business transactions are secured by using HTTPS and Digital Certificates.
Understanding Purchase

Slide Objective
To provide an overview of the topics in this section.

Lead-in
The topics covered in this section include...

- Overview of the Purchase Process
- Capture Payment Information
Once a shopper provides a ship-to address for the purchase, the checkout process enables the shopper to provide payment details such as:

- Credit card number
- Name on the card
- Expiration date
- Bill-to address

These details can be typed into an HTML form or by using Microsoft Wallet. When the shopper confirms the purchase, the purchase pipeline is executed to verify credit card information that the shopper has provided. If the information is valid, the pipeline saves the order, generates and displays a receipt.
Capture Payment Information

Slide Objective
To accept shopper’s credit card information by using HTML form or Microsoft Wallet.

Lead-in
Microsoft Wallet provides shoppers an easy and secure way to store address and payment information on the shopper’s computer.

Using HTML Forms
HTML forms are created by using HTML tags in .asp files. The payment.asp file contains the HTML code for accepting the shopper’s credit card information and bill-to address and posts this information to xt_orderform_purchase.asp.

The xt_orderform_purchase.asp file validates the shopper information and then executes the Purchase OPP as a transacted pipeline. You will learn more about a transacted pipeline later in this module.

Using Microsoft Wallet
Microsoft Wallet provides shoppers an easy and secure way to store payment and address information on the shopper’s computer. Microsoft Wallet consists of The Payment Selector and the Address Selector, which are available as plug-ins for Netscape Navigator, and as ActiveX® controls for Microsoft Internet Explorer.

The Payment Selector and the Address Selector provide an interface that an online shopper can use to securely store payment and address information, and supply that information to vendors. The Payment Selector includes support for major credit card types such as VISA, MasterCard, American Express, Discover, and JCB.

If the Wallet controls are implemented, shoppers need not type their name, address, and other information into HTML forms every time they shop, as they usually would, since data can be stored in Wallet controls for reuse.

When a site is created by using Commerce Server's Site Builder Wizard, the scripting necessary for the Wallet controls is automatically built into the site.
Executing the Purchase OPP

**Slide Objective**
To provide an overview of the topics in this section.

**Lead-in**
The topics covered in this section include...

- Overview of the Purchase OPP
- Stages of the Purchase OPP
The Purchase pipeline is executed after an order form has successfully passed through the Plan pipeline and the shopper has confirmed the purchase.

- The Purchase pipeline:
  - Verifies the shopper’s payment details.
  - Accepts the final purchase of an order form.
  - Writes the order to the site database.

The configuration of a Purchase pipeline is saved in purchase.pcf.

As stated previously, a Purchase pipeline is a transacted pipeline. A transacted pipeline consists of components that are designed and configured to support Microsoft Transaction Server (MTS) transactions. A MTS transaction is a unit of work that is performed as an atomic operation. The Commerce Server OPP supports MTS transactions through the MtsTxPipeline object. The MtsTxPipeline object executes a Purchase pipeline as shown in the following code:

```vbscript
Set pipeline = Server.CreateObject(“Commerce.MtsTxPipeline”) Call pipeline.LoadPipe(“<path and filename of the .pcf file>”) pipeline.Execute(1, orderForm, pipeContext, 0)
```

For the creation and execution of a pipeline, see i_util.asp in Appendix A.
The Purchase pipeline consists of three stages.

**Purchase Check stage**

The Purchase Check stage is used to verify a shopper’s bill-to address and credit card information. The following table describes the components stored in Purchase.pcf created by the Site Builder Wizard.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validate Bill_to</td>
<td>Verifies that each Bill_to field on the order form contains data.</td>
</tr>
<tr>
<td>Validate CC Info</td>
<td>Verifies that each credit card field on the order form contains data and that the site supports the credit card type.</td>
</tr>
<tr>
<td>Validate CCNumber</td>
<td>Verifies that the credit card date has not expired and that the number is properly formed.</td>
</tr>
</tbody>
</table>

The script in the Validate Bill_to and Validate CC Info components can be customized to perform additional validation on bill-to address and credit card fields.

**Payment stage**

The Payment stage is used to approve credit card payments. The DefaultPayment component of this stage sets an initial value in the _payment_auth_code name/value pair. The RequiredPayment component ensures that _payment_auth_code is not NULL.

**Accept stage**

The Accept stage handles the completed order, including initiating order tracking, generating purchase order and saving the order. The following table describes some of the components related to this stage.
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveReceipt</td>
<td>Saves all fields on the order form that have a corresponding column name in the database table.</td>
</tr>
<tr>
<td>SQLItemADO</td>
<td>Runs the SQL command for each item in the order, passing the specified order form fields as parameters to the SQL command.</td>
</tr>
<tr>
<td>SQLOrderADO</td>
<td>Runs the SQL command once for each order passing the specified order parameters to the SQL command.</td>
</tr>
</tbody>
</table>
◆ Tracking an Order

**Slide Objective**
To provide an overview of the topics in this section.

**Lead-in**
The topics covered in this section include...

- How to Generate a Simple Order Number?
- How to Track an Order?
- Practice: Generating a Simple Order Number
How to Generate a Simple Order Number?

As mentioned in Module 5, the order number, or the order ID, uniquely identifies an order. It is assigned during a shopper’s initial order activity and is used to track the order through completion. The order number is a 26-character Globally Unique Identifier (GUID) generated by Commerce Server. Because the length of the GUID makes it difficult to remember, most merchants prefer to use a shorter order number for the convenience of their shoppers. The following procedure uses the Five Lakes Publishing sample site as an example to generate order numbers in the format

<current month><current year><n>

where n is the running sequence number.

To create a short order number

1. Create a table named <Sitename>_orderid with an attribute named NewOrderID of type char(26).
2. Insert a row in the <Sitename>_orderid table to store the starting order number for the site.
3. In the OrderFormPurchase function of xt_orderform_purchase.asp file:
   a. Retrieve the NewOrderID value from the <Sitename>_orderid table into a variable named New_oid. The order number stored in New_oid will be used for the current shopping session.
   b. Extract the sequence number from New_oid and increment it by 1.
   c. Prefix the incremented sequence number with the current month and current year to get the order number to be used for the subsequent shopping session.
   d. Replace NewOrderID value in the <Sitename>_orderid table with the newly created order number.
   e. Replace the default order id in the OrderForm object with the order number in New_oid.
How to Track an Order?

On successful completion of a purchase transaction, Commerce Server generates a purchase confirmation. The purchase confirmation, displayed in the form of an HTML page, contains the order number that can be used by a shopper to check the status of an order.

A shopper may be interested in viewing receipt details or shipping details. For example, in the Five Lakes Publishing sample site, a shopper can return to the site to view the shipping status of an order.

The following code retrieves order details from <Sitename>_receipt table and sets the shipping status of an order:

```vbscript
' cmdTemp.CommandText = Replace(MSCSQueryMap.orderid.SQLCommand,":1", "Request("order_track_id")")
Set rsorderid = Server.CreateObject("ADODB.Recordset")
rsorderid.Open cmdTemp, , adOpenForwardOnly, , adLockReadOnly
if not rsorderid.EOF then
    if (now() - rsorderid("date_entered"))>2 then
        ship_status="Shipped"
    else
        ship_status="Not Shipped"
    end if
```
In this practice, you will replace the complex GUID order number generated by Commerce Server with a short order number. Your site database contains a table named FiveLakes_orderid to store the order number.

**Caution** The activities in this practice are a prerequisite for successful completion of lab 7.

**Insert a row in the FiveLakes_orderid table**

1. On the Start menu, point to Programs, Microsoft SQL Server 7.0, and then click Query Analyzer.

2. In the Connect to SQL Server dialog box, select the Use SQL Server authentication option button, type sa in the Login Name text box, leave the Password text box blank, and then click OK.

**Note** You do not need to select the Server name as it is selected by default.

3. In the New query dialog box, select the FiveLakes database from the DB drop-down list box.

4. Type the following SQL statement into the Query window to store the initial order number in the format `mmYYYYn`, where `mm` is the current month, `YYYY` is the current year, and `n` is a running sequence number starting with 1.

   ```sql
   update FiveLakes_orderid set NewOrderID="mmYYYYn"
   ```

   For example, if the current month is May 1999, the initial order number would be created using the following SQL statement:

   ```sql
   update FiveLakes_orderid set NewOrderID="0519991"
   ```
5. Press F5 to execute the query.
   The result should show “1 row(s) affected”.

6. Close the Query Analyzer.

► Modify the xt_orderform_purchase.asp file to generate a simple order number

1. In Microsoft Visual InterDev™ 6.0, open the FiveLakes project and then double-click xt_orderform_purchase.asp in the Project Explorer window.

2. In the OrderFormPurchase function, below the line OrderFormPurchase = null, add the following code to create a short order number:
   ```vbscript
   Set rsOrderID = Server.CreateObject("ADODB.Recordset")
   ...
   REM Extract Order ID value from the table
   cmdTemp.CommandText = "select NewOrderID from FiveLakes_orderid"
   set rsOrderID = cmdTemp.Execute
   New_oid=rsOrderID("NewOrderID")
   ...
   temp_val=Int(Mid(New_oid,7)) + Int("1")
   if len(month(now())) = 2 then
       next_order_id = month(now()) & year(now()) & " "
       CStr(temp_val)
   else
       next_order_id = "0" & month(now()) & year(now()) & " "
       CStr(temp_val)
   end if
   ...
   cmdTemp.CommandText="update FiveLakes_orderid set NewOrderID = :1"
   next_order_id = "" & next_order_id & ""
   cmdTemp.CommandText=Replace(cmdTemp.CommandText,"","
   next_order_id)
   set rsOrderID = cmdTemp.Execute
   ```

3. Scroll to the following line, which initializes an order form object.
   ```vbscript
   Set MSCSOrderForm = 
   UtilGetOrderForm(MSCSOrderFormStorage,created)
   ```

4. On the next line, add code to replace the default order_id in the order form object with the short order number retrieved from the database as follows:
   ```vbscript
   mscsOrderForm.order_id = New_oid
   ```

5. On the File menu, click Save xt_orderform_purchase.asp, and then click Close.
Display the country on the payment page

Note: If you have completed the practice and the lab in “Module 6: Checking Out,” you will need to modify the payment page to accept the shopper’s bill-to-country and store it in the OrderForm object.

1. Edit the payment.asp file.
2. Locate the following code:
   ```html
   VALUE="<%= mscsPage.HTMLEncode(mscsOrderForm.bill_to_zip) %>
   </TD>&nbsp;
   </TR>
   ```
3. On the next line, add the following code to display the bill-to-country, which is the country selected by the shopper on the shipping page:
   ```html
   <TR>
   <TD ALIGN="RIGHT">
   <B>Country:</B>
   </TD>
   <TD>
   <% sqlText = MCSQueryMap.country.SQLCommand
   cmdTemp.CommandText = sqlText
   Set rsCountry = Server.CreateObject("ADODB.Recordset")
   rsCountry.Open cmdTemp, , adOpenForwardOnly, , adLockReadonly %>
   <td align="left">
   <select name="bill_to_country" size="1">
   <%While not rsCountry.EOF %>
   <%=
   mscsPage.Option(rsCountry("country").value, ,
   mscsOrderForm.bill_to_country))%>
   <%=
   rsCountry("country").value %>
   <% rsCountry.MoveNext
   Wend %>
   </select VALUE="<%= mscsPage.HTMLEncode(mscsOrderForm.bill_to_country) %>
   %>
   </td>
   </TR>
   ```
4. On the File menu, click Save payment.asp, and then click Close.

View the short order number on the site
1. Start Microsoft Internet Explorer and type http://localhost/FiveLakes in the address bar of Internet Explorer.
2. Shop for a product on the site.
3. On the Purchase Confirmation page, notice the short order number.
Securing Business Transactions

Slide Objective
To provide an overview of the topics in this section.

Lead-in
The topics covered in this section include...

- Introducing HTTPS
- Introducing Digital Certificates
Introducing HTTPS

- ASP file receiving confidential data should be secured by SSL
- SSL secures transactions between client and server
- HTTPS protocol provides access to a secured page
- Page.SURL method generates a HTTPS URL

Shoppers using a Commerce Server site need assurance that passwords and credit card numbers are protected from unauthorized access. A shopper must get a secure connection to an ASP that processes confidential information. When a shopper submits credit card information by using a form, Secure Sockets Layer (SSL) should secure the ASP file that receives the form’s post data. SSL is a method of data encryption used to secure transactions between a client (the shopper’s browser) and Commerce Server. To receive a page that is secured by SSL, the browser must send a request by using HTTPS protocol as follows:

https://example.microsoft.com/

To generate a URL by using HTTPS, you can use the page.SURL method. By default, HTTPS is disabled in the site that you create by using the Site Foundation Wizard and can be enabled by setting the Site.Disable HTTPS value to zero (0). The following code, when invoked from ASP, enables HTTPS:

```
MSCSSite.Disable HTTPS = 0
```

Assuming that HTTPS is enabled by setting Site.Disable HTTPS value to zero, the following code will generate the URL https://localhost/<Sitename>/purchase.asp:

```
<FORM METHOD = “POST” ➔
ACTION="<%=mcsPage.SURL(“xt_orderform_purchase.asp”) ➔
%>"
```
Introducing Digital Certificates

To see the animation "Using Digital Certificates" see the accompanying CD-ROM.

For a shopper to get a secure connection to a Web site, the server hosting the site should have a digital certificate.

Digital certificates are electronic documents that authenticate users and entities on a network and are issued by a Certificate Authority. Along with the digital certificate, the Certificate Authority also issues public and private keys.

For authentication, Commerce Server sends the digital certificate to the client. If messages are to be sent to the client, the messages are encrypted by using a private key and the public key is sent along with the digital certificate. Once the client receives the message, the client uses the sender’s public key to decrypt the message.
Lab 7: Tracking Order Status

After completing this lab, you will be able to:

- Retrieve order details from your site database.
- Display shipping status and order details for a specific order number.
1. What is the advantage of using Microsoft Wallet over HTML forms?
   
   With the Wallet controls implemented, shoppers need not type their name, address, and other information every time they shop, as with HTML forms, since data can be stored in Wallet controls for reuse.

2. What is the advantage of a simple order tracking number?
   
   The default order number is a 26-character GUID generated by Commerce Server, which is difficult for shoppers to remember. A simple order number makes order tracking easier for the shopper.

3. What is needed to get a secure connection to a Web Site?
   
   To get a secure connection, the server hosting the Web Site should have a server certificate, also referred to as a digital certificate.

4. What is HTTPS?
   
   HTTPS is a protocol that provides access to a Web page secured by SSL.
Instructor Notes Module 8: Tracking Shopper Information

Introduction

This module provides students with the knowledge necessary to track a returning shopper.

After completing this module, students will be able to:

- Store and retrieve shopper information by using cookies.
- Store and retrieve shopper information by using a registration table.

Materials and Preparation

This section provides you with the materials and preparation needed to teach this module.

Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file, P08_1588.ppt
- Module 8, “Tracking Shopper Information”
- Lab 8, “Tracking Shoppers Using Registration Table”

Preparation

To prepare for this module, you should:

- Read all the materials for this module.
- View the relevant .asp files of the Five Lakes Publishing sample site.
- Review the relevant sections of the product documentation.
- Complete the practice and lab in this module.
Module Strategy

Use the following strategy to present this module:

- Using Cookies to Track Shoppers
  Explain the benefits of registering shoppers.
  Explain how shopper information is stored and retrieved from a cookie.
  Encourage students to create cookies and view the contents of the cookie.

- Using Registration Table to Track Shoppers
  Explain the advantage of using a database table, rather than a cookie, to
  store shopper information.
  Explain the need for having two SQLOrderADO components and explain
  how shopper information is inserted and updated in a database table by
  using the SQLOrderADO component.
Module 8: Tracking Shopper Information
Overview

Slide Objective
To provide an overview of the topics covered in this module.

Lead-in
In this module, you will learn to track a returning shopper by using cookies and a registration table.

Objectives
After completing this module, you will be able to:

- Store and retrieve shopper information by using cookies.
- Store and retrieve shopper information by using a registration table.
◆ Using Cookies to Track Shoppers

**Slide Objective**
To provide an overview of the topics in this section.

**Lead-in**
The topics covered in this section include…

- Why Track Shoppers?
- Setting a Cookie
- Retrieving Shopper Information from a Cookie
- Practice: Using Cookies to Track Shoppers
Why Track Shoppers?

On a shopper’s first visit to a site, the site can capture and save shopper information or in other words, register the shopper. Registering a shopper provides the following benefits:

- The site can display a customized catalog based on the previous orders placed by a shopper.
- The site can notify a shopper of the latest products in the site by sending e-mail.
- The shopper need not type address and payment information on each visit to the site.

Commerce Server supports registration of shoppers to keep track of returning shoppers. The Site Builder Wizard supports three models of shopper registration:

- None
- On Entry
- On Ordering

Shoppers can be registered when they enter a site or when they add items to the shopping cart. When a shopper registers, the shopper information is saved in a database table.

On the sites that require registration, on each new visit to the site, the shopper signs in and the shopper’s information is retrieved from the database and associated with a new shopper_ID for the shopping session.
Setting a Cookie

As mentioned in Module 4, a cookie is a file that is created on a shopper’s computer. It is stored under the shopper’s login profile on the hard disk.

Information can be read from a cookie only if the cookie is set. Using the Response.Cookies method of ASP, you can set values in a cookie as follows:

```vbscript
‘%Response.Cookies("<Cookiename>")”("name")= mscsOrderForm.ship_to_name %>
```

Information stored in a cookie is lost when the shoppers quit their Web browser. This sort of cookie is referred to as a session cookie. In order to retain cookie information through multiple shopping sessions, you must set the Expires property of the cookie to a future date as follows:

```vbscript
REM--To set the Expires property of the cookie
‘% Response.Cookies("<Cookiename>").Expires#<Month> <DD>,#<YYYY>%
```

Note In an .asp file, the Response and Request methods of the cookies collection must be executed prior to executing any HTML code.
Retrieving Shopper Information from a Cookie

Using the **Request.Cookies** method of ASP, you can retrieve and display shopper information stored in a cookie as follows:

```javascript
<% mscsOrderForm.ship_to_name= Request.Cookies("<Cookiename>")("name") %>
<% mscsOrderForm.ship_to_street= Request.Cookies("<Cookiename>")("street") %>
```

These details can now be modified and written back into the cookie by using the **Response.Cookies** method.
In this practice, you will retrieve and store shopper information by using a cookie.

**Configure a site to use URLs instead of cookies**

1. In Microsoft® Visual InterDev™ 6.0, open the FiveLakes project, and then open global.asa.
2. Locate the following line of code:
   ```
   call MSCSShopperManager.InitManager(vRoot,"cookieurl")
   ```
3. Change this code to:
   ```
   call MSCSShopperManager.InitManager(vRoot,"url")
   ```
4. Locate the following line of code:
   ```
   Set Application("MSCSShopperManager")=MSCSShopperManager
   ```
5. On the next line, add the following code to initialize an Application MSCSSIDUrlKey variable. This provides a name to be used for the query string variable to be appended to the URL.
   ```
   Application("MSCSSIDUrlKey") = "mscssid"
   ```
6. Save and close global.asa.
7. Open the i_shop.asp file.
8. Locate and delete the following code in i_shop.asp:
   ```
   call Response.Redirect(pageURL("default.asp"))
   ```
9. Save and close i_shop.asp.
Store shopper information in a cookie

1. Edit the payment.asp file.

2. Before the <html> tag, insert the following code to set the expiration date of the cookie to a future date:

   ```vbscript
   <% Response.Cookies("<Cookiename>").Expires=#<Month>#, <DD>, <YYYY># %>
   ```

   **Note**  
   `<Cookiename>` should be changed to some useful string without the brackets. Also, the month should be a text string. Finally, the day and year information should be numerals, all set at some time in the future.

3. On the next line, add code to store the shopper information in the cookie as follows:

   ```vbscript
   <%Response.Cookies("<Cookiename>")("name")= mscsOrderForm.ship_to_name %>
   <%Response.Cookies("<Cookiename>")("street")= mscsOrderForm.ship_to_street %>
   <%Response.Cookies("<Cookiename>")("city")= mscsOrderForm.ship_to_city %>
   <%Response.Cookies("<Cookiename>")("state")= mscsOrderForm.ship_to_state %>
   <%Response.Cookies("<Cookiename>")("country")= mscsOrderForm.ship_to_country %>
   <%Response.Cookies("<Cookiename>")("zip")= mscsOrderForm.ship_to_zip %>
   <%Response.Cookies("<Cookiename>")("phone")= mscsOrderForm.ship_to_phone %>
   ```

4. On the File menu, click **Save payment.asp**, and then click **Close**.

Retrieve shopper information from a cookie

1. In the Project Explorer window, double-click shipping.asp.

2. Insert the following code before the <html> tag in shipping.asp to retrieve shopper information from the cookie and store it in the OrderForm:

   ```vbscript
   <% mscsOrderForm.ship_to_name= Request.Cookies("<Cookiename>")("name") %>
   <% mscsOrderForm.ship_to_street= Request.Cookies("<Cookiename>")("street") %>
   <% mscsOrderForm.ship_to_city= Request.Cookies("<Cookiename>")("city") %>
   <% mscsOrderForm.ship_to_state= Request.Cookies("<Cookiename>")("state") %>
   <% mscsOrderForm.ship_to_country= Request.Cookies("<Cookiename>")("country") %>
   <% mscsOrderForm.ship_to_zip= Request.Cookies("<Cookiename>")("zip") %>
   <% mscsOrderForm.ship_to_phone= Request.Cookies("<Cookiename>")("phone") %>
   ```

3. On the File menu, click **Save shipping.asp**, and then click **Close**.
View shopper information retrieved from a cookie

1. Start Microsoft Internet Explorer and type in the URL address
http://localhost/FiveLakes.
2. Shop at the site and on the Shipping page, click Click here if you have problems with the wallet to view the HTML form.
3. Type your shipping information in the HTML form as a new shopper and click Total. The script in payment.asp will save the shipping details in a cookie.
4. On the Final Purchase Approval page, type in the credit card information and click Purchase to complete the purchase.
5. Return to the site and view your shipping information on the shipping page. The script in shipping.asp has successfully retrieved your shipping details from the cookie.
◆ Using Registration Table to Track Shoppers

Slide Objective
To provide an overview of the topics in this section.

Lead-in
The topics covered in this section include…

- Creating a Shopper Table
- Retrieving Shopper Information
- Modifying the Purchase Pipeline
Creating a Shopper Table

Storing shopper information in a table is more secure than storing it in a cookie. This is because a cookie can be accidentally deleted from the shopper's computer. Also, a database table is preferred over a cookie since the shopper's Web browser may not always support cookies.

Saving shopper information in a table

Commerce Server creates a <Sitename>_shopper table on the site database for the sites that require shopper registration. The <Sitename>_shopper table contains name, street, city, state, zip, country, phone, e-mail, and shopper_ID attributes.

On the sites that are created with the registration option set to None in the Site Builder Wizard, such as the Five Lakes Publishing sample site, you can add the shopper registration functionality at a later stage.

Adding shopper registration information to a site

In order to include the shopper registration functionality in your site, you need to add the following elements:

- A table in the site database to store shopper information.
- A query description in global.asa to retrieve shopper information.
- A component in the Purchase pipeline to update shopper information in the site database, for returning shoppers.
- A component in the Purchase pipeline to insert shopper information in the site database, for new shoppers.

To create a shopper table in the site database

1. On the Start menu, point to Programs, Microsoft SQL Server 7.0, and then click Query Analyzer.
2. Log on to the SQL Server.
3. Execute the query to create a table as follows:

```sql
create table <Sitename>_shopper
    (shopper_ID varchar(64) constraint Prime_Key PRIMARY KEY,
     name varchar(30),
     city varchar(20),
     state varchar(20),
     zip varchar(10),
     country varchar(30),
     phone varchar(20),
     email varchar(30))
```
Retrieving Shopper Information

Shopper information can be captured and retrieved from `<Sitename>_shopper` table when:

- A shopper enters the site.
- A shopper types in shipping details.

In the Five Lakes Publishing sample site, the script in `shipping.asp` accepts shopper information for a new shopper and displays shopper information for a returning shopper.

**To retrieve returning shopper’s information from the site database**

1. Add a query in `global.asa` to retrieve shopper information from the `<Sitename>_shopper` table as follows:

   ```vbscript
   Set MSCSQueryMap.email_search = AddQuery(""
   SELECT * FROM `<Sitename>_shopper`
   WHERE email = '1'")
   ```

2. In Microsoft Visual InterDev 6.0, modify the script in `shipping.asp` to:
   a. Accept the returning shopper’s e-mail address by using HTML tags.
   b. Execute the query to retrieve shopper information from the `<Sitename>_shopper` table as follows:

   ```vbscript
   cmdTemp.CommandText = "SELECT * FROM FiveLakes_shopper"
   WHERE email = '1' & email & "'

   Set rsShopperFindSpec = Server.CreateObject("ADODB.Recordset")
   rsShopperFindSpec.Open cmdTemp, , adOpenStatic, , adLockReadOnly
   ```
3. Store the shopper information in the **OrderForm** object as follows:

   REM-If the **email** exists in the table
   if not rsShopperFindSpec.EOF then
   mscsOrderForm.ship_to_email = rsShopperFindSpec("email")

4. Display the **OrderForm** values by using HTML tags.

   For the complete code of shipping.asp created in the Five Lakes Publishing sample site, refer to Appendix A.
Once a shopper confirms a purchase, the shopper information is posted to the xt_orderform_purchase.asp file where the Purchase pipeline is executed and the shopper's information is committed to the site database.

Two SQLOrderADO components are required to insert and update shopper information in the site database for the new and returning shoppers, respectively.

Since the Purchase pipeline is a transacted pipeline that requires either all or none of the components to be executed, the two SQLOrderADO components cannot be added to the same pipeline. This is because, at any point in time, either an update operation for a returning shopper or an insert operation for a new shopper should occur.

Inserting information to the site database

The Five Lakes Publishing sample site contains two Purchase pipelines, each having a SQLOrderADO component. The SQLOrderADO component in the first pipeline inserts shopper information in the <Sitename>_shopper table for a new shopper as follows:

```sql
insert into <Sitename>_shopper(email,name,street,city,state,country, zip,phone) values (?,?,?,?,? ,?,?,?)
```

with the following parameter list:

```sql
order.ship_to_email order.ship_to_name order.ship_to_street order.ship_to_city order.ship_to_state order.ship_to_country order.ship_to_zip order.ship_to_phone
```
The SQLOrderADO component in the second pipeline updates shopper information in the `<Sitename>_shopper` table for a returning shopper as follows:

```
update <Sitename>_shopper set name=?, street=?, city=?, state=?, country=?, zip=?, phone=? where email=?
```

with the following parameter list:

```
order.ship_to_name order.ship_to_street order.ship_to_city  
order.ship_to_state order.ship_to_country  
order.ship_to_zip order.ship_to_phone order.ship_to_email
```
The script in xt_orderform_purchase.asp is then modified to execute the appropriate pipeline.

REM--Execute the query in global.asa to determine whether the shopper is a new shopper or a returning shopper.
cmdTemp.CommandText = Replace(MCSQueryMap.email_search.CommandText,":\\1", Replace(mscsOrderForm.ship_to_email, "\\"", " ""))
Set rsEmail = Server.CreateObject("ADODB.Recordset")
rsEmail.Open cmdTemp, , adOpenKeyset, adLockReadOnly

REM--If the shopper is a returning shopper
if not rsEmail.EOF then
    errorLevel = UtilRunTxPipe("purchaseupdatepcf", mscsOrderForm, mscsPipeContext)
else
    REM--If the shopper is a new shopper
    errorLevel = UtilRunTxPipe("purchaseinsertpcf", mscsOrderForm, mscsPipeContext)
end if

Note UtilRunTxPipe is a function in i_util.asp that contains the code to execute the pipeline. The i_util.asp file is included in xt_orderform_purchase.asp.
Lab 8: Using Registration Table to Track Shoppers

After completing this lab, you will be able to:

- Retrieve and update shopper information for a returning shopper.
- Insert shopper information, for a new shopper, in the table.
- View shopper information retrieved from a table.
1. What is the advantage of registering shoppers on your site?
   
   If a registration table or cookies are implemented, shoppers need not type their name, address, and other information every time they shop. Shopper information is stored in a registration table or cookie and can be retrieved when the shopper returns.

2. What are the three models of shopper registration supported by the Site Builder Wizard?
   
   None, On Ordering, and On Entry are the three models of shopper registration supported by the Site Builder Wizard.

3. Which method do you use to set a cookie?
   
   The Response.Cookies method is used to set a cookie.

4. Explain the advantage of using a database table, instead of a cookie, to store shopper information.
   
   A database table is stored on the server and, therefore, is more secure than a cookie, which is stored on the client computer and can be accidentally deleted. Also, a shopper’s Web browser may not always support cookies.
Instructor Notes Module 9: Introducing Business-to-Business Commerce

Introduction

Presentation:
15 Minutes

Lab:
45 Minutes

This module provides students with the knowledge necessary to implement business partner functionality in their site.

After completing this module, students will be able to:

- Explain business-to-business commerce.
- Implement business partner functionality in their site.

Materials and Preparation

This section provides you with the materials and preparation needed to teach this module.

Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file P09_1588.ppt
- Module 9, “Introducing Business-to-Business Commerce”
- Lab 9, “Implementing Business Partner Functionality”

Preparation

To prepare for this module, you should:

- Read all the materials for this module.
- View the relevant .asp files of the Five Lakes Publishing sample site.
- Complete the lab for this module.
Module Strategy

Use the following strategy to present this module:

- **Business-to-Business Commerce**
  Explain what is business-to-business commerce. Explain the role of Commerce Interchange Pipeline (CIP) in business-to-business commerce. Discuss the concept and implementation of ePartners. Encourage students to visualize some business-to-business scenarios.

- **Business Partner Functionality**
  Describe the code in the order_epartner.asp and i_epartner.asp files, step by step, to explain the implementation of business-to-business commerce in the Five Lakes Publishing sample site. In the Site Manager page, display the **Orders by ePartner** report.
Module 9: Introducing Business-to-Business Commerce
Overview

**Slide Objective**
To provide an overview of the module topics and objectives.

**Lead-in**
In this module, you will learn about business-to-business commerce and how the business partner functionality can be implemented.

- Business-to-Business Commerce
- Business Partner Functionality
- Lab 9: Implementing Business Partner Functionality (Optional)
- Review

After completing this module, you will be able to:

- Explain business-to-business commerce.
- Implement business partner functionality in your site.
Business-to-Business Commerce

Slide Objective
To provide an overview of the topics in this section.

Lead-in
The topics covered in this section include...

- Overview of Business-to-Business Commerce
- A Business-to-Business Scenario
Overview of Business-to-Business Commerce

Business-to-business commerce includes online wholesaling, in which businesses sell goods and services to other businesses over the Web. Enabling businesses to exchange information electronically is critical to the success of a business-to-business commerce site. The Commerce Interchange Pipeline (CIP) enables businesses of all sizes to exchange information electronically.

You can use the CIP to securely transmit business data objects between business partners. A business data object is defined as a data container for business information such as:

- Purchase orders
- Purchase order receipts
- Sale notices
- Invoices
- Billing records

Creating a Commerce Interchange Pipeline

A CIP is created in the same way that an OPP is created.

A business-to-business scenario often involves two CIPs running on separate servers. One pipeline packages information and transmits it over the network and the second pipeline reads the information from the network.

Commerce Server is a platform that facilitates the development of Web-based commerce sites for business-to-business transactions.
A Business-to-Business Scenario

The implementation of the business-to-business commerce on the Five Lakes Publishing sample site is as follows:

Five Lakes Publishing advertises its Web site through other popular sites. The incentive for the owners of other Web sites to advertise the Five Lakes Publishing site is a commission on every sale referred through their sites. These Web-site owners are called business partners, or ePartners. Every ePartner is given a unique ID. In addition, every order has an ePartner ID associated with it. This enables Five Lakes Publishing to compute commissions for its trading partners.
Business Partner Functionality

- Storing ePartner Information
- Verifying an ePartner
- Associating ePartner ID with an Order
- Displaying Orders Received through ePartners
To implement business partner functionality, the sample site includes the following elements:

- A table in the site database to store ePartner information.
- An epartner_ID attribute in the receipt table to associate every order with an ePartner.
- A query description in global.asa to retrieve ePartner information.
- An .asp file to retrieve epartner_ID from the URL and verify its existence in the ePartner table.
- An .asp file to display orders received through ePartners.

Five Lakes Publishing maintains a table with the following attributes to store information about its ePartners:

- epartner_ID
- name
- site_address
Verifying an ePartner

Slide Objective
To verify the existence of ePartnerID in the site database.

Lead-in
The site database must contain information of all the authorized business partners of your site.

- The global.asa file contains a query to retrieve ePartner information from FiveLakes_epartner table.
- The i_epartner.asp file contains a script to verify the existence of an ePartner.

When a shopper enters the Five Lakes Publishing sample site from an ePartner’s site, the ePartner ID is passed through the URL into the sample site. This ePartner must have a valid entry in the FiveLakes_epartner table. In other words, the ePartner information must exist in the ePartner table of the sample site or the referring site will not be credited.

In the Five Lakes Publishing sample site, the script in i_epartner.asp retrieves the epartner_ID from the URL and checks for the existence of a business partner in the FiveLakes_epartner table. The i_epartner.asp file is included in every page a shopper might visit.

In the sample site:
- The global.asa file contains a query to retrieve ePartner information from the FiveLakes_epartner table as follows:

```vbscript
Set MSCSQueryMap.epartner = AddQuery(""
SELECT epartner_ID 
FROM FiveLakes_epartner 
WHERE epartner_id = ':1'")
```
The i_epartner.asp file contains a script to verify the existence of an ePartner as follows:

```plaintext
epartner_id=0
epartner_id=Request.QueryString("epartner_id")
if epartner_id=0 then
  epartner_id=1
else
  cmdTemp.CommandText = Replace(MSCSQueryMap.epartner.SQLCommand,":1", epartner_id)
  Set rsepartner = Server.CreateObject("ADODB.Recordset")
  rsepartner.Open cmdTemp, , adOpenForwardOnly, adLockReadOnly
  if rsepartner.EOF then
    epartner_id=1
  end if
end if
end if %>
```
Every order is stored along with the ePartner ID in the FiveLakes_receipt table. This helps the site to compute the commission for an ePartner based on the number of orders and the total order value.

Orders that are not received through an ePartner are assigned the epartner_ID value of 1 in the FiveLakes_receipt table.

In the Five Lakes Publishing sample site:

- The FiveLakes_receipt table contains an attribute epartner_id.
- The xt_orderform_prepare.asp file contains a script to store ePartner ID in the OrderForm object.
- The SaveReceipt component in Accept stage of Purchase pipeline stores the OrderForm values in FiveLakes_receipt table.
Displaying Orders Received through ePartners

- Orders received through ePartners can be viewed from the Site Manager page
- In the Five Lakes Publishing sample site, order_epartner.asp retrieves and displays these orders

The Five Lakes Publishing sample site enables site operators and site administrators to view orders received through ePartners. The site contains an Orders by ePartner report that can be viewed from the Site Manager page. This report displays the following information for every order in the FiveLakes_receipt table:

- ePartner ID
- Order ID
- Order Date
- Number of items
- Order total

In the sample site, the order_epartner.asp file contains the script to retrieve and display order information from the FiveLakes_receipt table.

For the complete code of order_epartner.asp, see Appendix A.
Lab 9: Implementing Business Partner Functionality (Optional)

After completing this lab, you will be able to:

- Create a database table to store the ePartner information.
- Create an .asp file to validate the ePartner ID passed in the URL.
- Create an .asp file to display orders received through ePartners.
1. What is business-to-business commerce?

   Business-to-business commerce includes online wholesaling, in which businesses sell goods and services to other businesses on the Web.

2. What is Commerce Interchange Pipeline?

   Commerce Interchange Pipeline is used to securely transmit business data objects between business partners.

3. How can you view the orders received through ePartners?

   The orders received through ePartners can be viewed from the Site Manager page.
Appendix A: Lab and Practice Solution Files

This appendix contains printed versions of all the files that students work with during the course, as well as number of other important files in the sample site.

These files are presented just as the Commerce Server wizards created them, without making changes to enhance readability.

All of these files were taken from the Solution folder for Chapter 9.

Table of Contents

<table>
<thead>
<tr>
<th>File Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>219</td>
</tr>
<tr>
<td>basket.asp</td>
<td>221</td>
</tr>
<tr>
<td>confirmed.asp</td>
<td>227</td>
</tr>
<tr>
<td>Default.asp</td>
<td>228</td>
</tr>
<tr>
<td>dept.asp</td>
<td>230</td>
</tr>
<tr>
<td>find.asp</td>
<td>232</td>
</tr>
<tr>
<td>global.asa</td>
<td>235</td>
</tr>
<tr>
<td>i_epartner.asp</td>
<td>241</td>
</tr>
<tr>
<td>i_error.asp</td>
<td>242</td>
</tr>
<tr>
<td>i_footer.asp</td>
<td>243</td>
</tr>
<tr>
<td>i_header.asp</td>
<td>244</td>
</tr>
<tr>
<td>i_mswallet.asp</td>
<td>245</td>
</tr>
<tr>
<td>i_selector.asp</td>
<td>246</td>
</tr>
<tr>
<td>i_shop.asp</td>
<td>256</td>
</tr>
<tr>
<td>i_util.asp</td>
<td>258</td>
</tr>
<tr>
<td>order.asp</td>
<td>261</td>
</tr>
<tr>
<td>order_ePartner.asp</td>
<td>262</td>
</tr>
<tr>
<td>order_status.asp</td>
<td>264</td>
</tr>
<tr>
<td>ordertrack.asp</td>
<td>265</td>
</tr>
<tr>
<td>payment.asp</td>
<td>266</td>
</tr>
<tr>
<td>product.asp</td>
<td>275</td>
</tr>
<tr>
<td>product_alt.asp</td>
<td>279</td>
</tr>
<tr>
<td>product_edit.asp</td>
<td>283</td>
</tr>
<tr>
<td>receipt.asp</td>
<td>291</td>
</tr>
<tr>
<td>receipts.asp</td>
<td>296</td>
</tr>
<tr>
<td>shipping.asp</td>
<td>298</td>
</tr>
<tr>
<td>xt_orderform_additem.asp</td>
<td>299</td>
</tr>
<tr>
<td>xt_orderform_clearitems.asp</td>
<td>301</td>
</tr>
</tbody>
</table>
xt_orderform_delitem.asp .................................................................302
xt_orderform_edititem.asp ............................................................303
xt_orderform_editquantities.asp ....................................................305
xt_orderform_prepare.asp ............................................................306
xt_orderform_purchase.asp ............................................................309
basket.asp

```vbscript
<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>
<% Response.ExpiresAbsolute=DateAdd("yyyy", -10, Date) %>
<---# INCLUDE FILE="i_shop.asp" -->
<---# INCLUDE FILE="i_ePartner.asp" -->
<---# INCLUDE FILE="i_util.asp" -->

<% REM Run the basic plan %>
Set mscsOrderForm = UtilRunPlan()
Set orderFormItems = mscsOrderForm.Items
nOrderFormItems = orderFormItems.Count
Set mscsBasketErrors = mscsOrderForm._Basket_Errors
nBasketErrors = mscsBasketErrors.Count
%

<HTML>
<HEAD>
<TITLE><%= displayName %>: Basket</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>
<BODY
BGCOLOR="#FFFFFF"
TEXT="#000000"
LINK="#FF0000"
VLINK="#FF0000"
ALINK="#FF0000"
>
<---# INCLUDE FILE="i_header.asp" -->

<H1>Shopping Basket</H1>
<% if Request("error").Count <> 0 then %>
<TABLE WIDTH="500">
<TR><TD><FONT COLOR="#FF0000"><STRONG>Quantity must be a number less than 1000.</STRONG></TD></TR>
</TABLE>
</p>
<% end if %>
<% if nOrderFormItems = 0 then %>
<BLOCKQUOTE>

</BLOCKQUOTE>
```
Your basket is empty.

If nBasketErrors > 0 then

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No errors</td>
</tr>
<tr>
<td>1</td>
<td>Error 1</td>
</tr>
<tr>
<td>2</td>
<td>Error 2</td>
</tr>
</tbody>
</table>

You have 0 items in your shopping basket.

To change an item's quantity, edit the number and press "Update Basket".

Purchase

Clear Order
<table>
<thead>
<tr>
<th>Label</th>
<th>Name</th>
<th>Unit Price</th>
<th>Today's Price</th>
<th>Qty</th>
<th>Extra Disc.</th>
<th>Total Price</th>
</tr>
</thead>
</table>

For more information, please refer to the Appendix A: Lab and Practice Solution Files.
<TR>
    <% for iLineItem = 0 to nOrderFormItems - 1 %>
    <% set lineItem = orderFormItems(iLineItem) %>
    <TD VALIGN="TOP">
        <%= mscsPage.HTMLEncode(lineItem.sku) %>
    </TD>
    <TD VALIGN="TOP">
        <%= mscsPage.HTMLEncode(lineItem._product_name) %>
    </TD>
    <TD VALIGN="TOP" ALIGN="RIGHT">
        <%= MSCSDataFunctions.Money(lineItem._product_list_price) %>
    </TD>
    <TD VALIGN="TOP" ALIGN="RIGHT">
        <%= MSCSDataFunctions.Money(lineItem._iadjust_currentprice) %>
    </TD>
    <TD WIDTH="60" VALIGN="TOP">
        <INPUT TYPE="Text" NAME="<%= mscsPage.HTMLEncode("qty_" & iLineItem) %>", SIZE=3,1 VALUE="<%= lineItem.quantity %>">
    </TD>
    <TD WIDTH="60" VALIGN="TOP" ALIGN="RIGHT">
        <%= MSCSDataFunctions.Money(lineItem._oadjust_discount) %>
    </TD>
    <TD VALIGN="TOP" ALIGN="RIGHT">
        <%= MSCSDataFunctions.Money(lineItem._oadjust_adjustedprice) %>
    </TD>
    <TD VALIGN="TOP">
        <A HREF="<%= baseSURL("xt_orderform_delitem.asp") & ">
            &epartner_id=" & epartner_id & "&" & mscsPage.URLShopperArgs("index", iLineItem) %>
        <IMG SRC="<%= "/" & siteRoot %>/manager/MSCS_Images/buttons/btnremove1.gif" BORDER="0" ALT="Delete item">
    </TD>
    </TR>
</TR>

REM -- Create Recordset, execute Query
cmdTemp.CommandText = Replace(MSCSQueryMap.related_products_upsell.SQLCommand, ":1", Replace(lineitem.sku, "", ","))
set rsUpsell = Server.CreateObject("ADODB.Recordset")
rsUpsell.Open cmdTemp, , adOpenKeyset, adLockReadOnly
if rsUpsell.recordcount >0 then
    while not rsUpsell.EOF
        set checkitem=mscsOrderForm.items
        inbasket =0
        for each row_checkitem in checkitem
            if row_checkitem.sku=rsUpsell("Upsell_sku").value then
                inbasket =1
                end if
        next
        if Cbool(inbasket=0) then
            <tr>
            <td colspan="1">&nbsp;&nbsp;</td>
            <td colspan="5"><a HREF="<%=mscsPage.URL("product_alt.asp","sku",rsUpsell("Upsell_sku").value,"index",ilineitem"quantity",ilineitem.quantity,"dept_id",ilineitem[_product_dept_id])%>"***<%=rsUpsell("description")%>***"</a></td>
            </tr>
        <%end if
        rsUpsell.MoveNext
    Wend
    end if %>
    <% next %>
    <% REM show subtotal: %>
    <TR>
    <TH BGCOLOR="#000000" COLSPAN="6" ALIGN="RIGHT"><FONT COLOR="#FFFFFF">Subtotal:</FONT></TH>
    <TD ALIGN="RIGHT"><%=MSCSDataFunctions.Money(mscsOrderForm._oadjust_subtotal)%></TD>
    </TR>
</FORM>
</TABLE>
<% end if %>
confirmed.asp

```vbscript
<!--# LANGUAGE=vbscript enable session state=false LCID=1033 -->

<html>
<head>
<title><%= displayName %>: Purchase Confirmation</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head>
<body bgcolor="#FFFFFF" text="#000000" link="#FF0000" vlink="#FF0000" alink="#FF0000">
<!--#include file="i_header.asp" -->
<h1>Purchase Confirmation</h1>
<p>Your order number is
<%= order_id = mscsPage.HTMLEncode(Request("order_id")) %>
</p>

Please record it for referencing your order.

<p>Thank you for your shopping at <%= displayName %>.</p>

<p>If you want to continue shopping, simply return to the <a href=""><% = pageURL("default.asp") & amp; epartner_id=" & amp; epartner_id %>">lobby</a>.</p>

<!--#include file="i_footer.asp" -->
</body>
</html>
```
Default.asp

```html
<!--#INCLUDE FILE="i_shop.asp" -->
<!--#INCLUDE FILE="i_ePartner.asp" -->
<HTML>

<HEAD>
  <TITLE><%= displayName %>: Lobby</TITLE>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>

<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#FF0000" VLINK="#FF0000" ALINK="#FF0000">

<!--#INCLUDE FILE="i_header.asp" -->

<H1><%= displayName %></H1>

<P>
Welcome to our shop. We have a broad range of products you can choose from.

<% sqlText = MSCSQueryMap.depts.SQLCommand %>
Set rsDepts = MSCS.Execute (sqlText, nDepts, adCmdText)

if rsDepts.EOF then
  %>
  <P>
  There are currently no departments available.
  </p>
else %>
  <P>
  Select a department below:
  </p>
  <UL>
    <LI><%= dept_nameField %></LI>
</UL>
  <LI><%= dept_idField %></LI>
</UL>
```

```
  %>
  <A HREF="depts.asp" & mcsPage.URLShopperArgs("dept_id", dept_idField, "ePartner_id", ePartner_id )" %>
  %>
</A>
```
You can see your order history on the <A HREF="<%=pageURL("receipts.asp") & "&partner_id=" &partner_id%>Order History Page</A>.

You can see your order status on the <A HREF="<%=pageURL("ordertrack.asp") & "&partner_id=" &partner_id%>Track Your Order Page</A>.

<!--#INCLUDE FILE="i_footer.asp" -->

</BODY>

</HTML>
dept.asp

```vbscript
<%@ LANGUAGE=vbscript enabelsessionstate=false LCID=1033 %>

<!--# INCLUDE FILE="i_shop.asp" -->
<!--# INCLUDE FILE="i_ePartner.asp" -->
<%
dept_id = mscsPage.RequestNumber("dept_id", "0")
sqlText = Replace(MSCSQueryMap.dept_by_id.SQLCommand, ":1", dept_id)
Set rsDept = MSCS.Execute(sqlText, nDepts, adCmdText)
if rsDept.EOF then
  dept_exists = false
  dept_name = "Unknown Department"
else
  dept_exists = true
  dept_name = rsDept("dept_name").value
  dept_description = rsDept("dept_description").value
  Response.AppendToLog "&" & mscsPage.URLArgs("MSS.Request.Category Name", dept_name)
end if
rsDept.Close
%

<HTML>

<HEAD>
  <TITLE><%= displayName %>: Department: '<%= mscsPage.HTMLEncode(dept_name) %>'</TITLE>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>

<BODY
  BGCOLOR="#FFFFFF"
  TEXT="#000000"
  LINK="#FF0000"
  VLINK="#FF0000"
  ALINK="#FF0000"
>
<%
if dept_exists then
cmdTemp.CommandText = Replace(MSCSQueryMap.products_by_dept.SQLCommand, ":1", dept_id)
Set rsProducts = Server.CreateObject("ADODB.Recordset")
rsProducts.Open cmdTemp, , adOpenStatic, adLockReadOnly
```
if rsProducts.EOF then
  products_exist = false
else
  products_exist = true
end if
end if

<!--#INCLUDE FILE="i_header.asp" -->

<%
  if not dept_exists then %>
    <p>The department you requested is currently not available.</p>
  <% else %>
    <h1><%= mscsPage.HTMLEncode(dept_name) %></h1>
    <p><%= mscsPage.HTMLEncode(dept_description) %></p>
  <% end if %>
<% if products_exist then %>
  Select a product from the list below:

  <ul>
    <% set skuField = rsProducts("sku")
      set nameField = rsProducts("name")
      do while Not rsProducts.EOF
        <li><a href="<%= baseURL("product.asp") & mscsPage.URLShopperArgs("dept_id", dept_id, "sku", skuField.value) & "&partner_id=" & epartner_id %>"{%= mscsPage.HTMLEncode(nameField.value) %}></a></li>
        <% rsProducts.MoveNext
        loop
      rsProducts.Close
    </ul>
  <% end if %>
<% end if %>

<!--#INCLUDE FILE="i_footer.asp" -->

</body>
</html>
find.asp

<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<!--# INCLUDE FILE="i_shop.asp" -->
<!--# INCLUDE FILE="i_ePartner.asp" -->
<HTML>

<HEAD>
   <TITLE><%= displayName %>: Find</TITLE>
   <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>

<BODY
   BGCOLOR="#FFFFFF"
   TEXT="#000000"
   LINK="#FF0000"
   VLINK="#FF0000"
   ALINK="#FF0000">
   <!--# INCLUDE FILE="i_header.asp" -->

<H1>Find</H1>

<!--# INCLUDE FILE="i_footer.asp" -->
</BODY>

<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<!--# INCLUDE FILE="i_footer.asp" -->

<!--# INCLUDE FILE="i_header.asp" -->

<!--# INCLUDE FILE="i_shop.asp" -->
<!--# INCLUDE FILE="i_ePartner.asp" -->
<FORM NAME="find_spec" METHOD="POST" ACTION="pageURL("find.asp") & "epartner_id" & epartner_id">
   Find:
   <%
   search_by = trim(mscsPage.RequestString("sselect"))
   strFindSpec = mscsPage.RequestString("find_spec")
   %>
   <td width="125%" colspan="2"><select name="sselect" size="1" tabindex="20">
      <%= mscsPage.Option("Keyword", search_by) %>
      <%= mscsPage.Option("Category", search_by) %>
   </select></td>
   <INPUT TYPE="TEXT" NAME="find_spec" SIZE="32" MAXLENGTH="200" VALUE="<%= mscsPage.HTMLEncode(strFindSpec) %>">
   <INPUT TYPE="SUBMIT" VALUE="Find">
</FORM>

<%
   if Request("find_spec").Count = 0 then
      hasFindSpec = false
      strFindSpec = ""
      nProductsFindSpec = 0
   else
      hasFindSpec = true
      strFindSpec = mscsPage.RequestString("find_spec")
   %>

<%}
REM -- escape wildcard characters and quote: ', _, %
if IsNull(strFindSpec) then
    strFindSpec = ""
    nProductsFindSpec = 0
else
    safeFindSpec = Replace(Replace(Replace(strFindSpec,"'","''"),"_","[_]"),"%","[ % ]")
end if
if search_by = "Keyword" then
    sqlText = MSCSQueryMap.find_by_keyword.SQLCommand
elseif search_by = "Category" then
    sqlText = MSCSQueryMap.find_by_category.SQLCommand
end if
sqlText = Replace(sqlText,":1",safeFindSpec)
cmdTemp.CommandText = sqlText
Set rsProductsFindSpec = Server.CreateObject("ADODB.Recordset")
    rsProductsFindSpec.Open cmdTemp, , adOpenStatic, adLockReadOnly
nProductsFindSpec = 0
Do While Not rsProductsFindSpec.EOF
    nProductsFindSpec = nProductsFindSpec + 1
    rsProductsFindSpec.MoveNext
Loop
if Not rsProductsFindSpec.BOF then
    rsProductsFindSpec.MoveFirst
end if
end if
if hasFindSpec then
    <p>
    <% if strFindSpec = "" then %>
        You must enter a find string before clicking Find.
    <% elseif nProductsFindSpec = 0 then %>
        There are no products that contain "<%= strFindSpec %>".
    <% else %>
        <% if nProductsFindSpec = 1 then %>
            There is 1 product that contains "<%= strFindSpec %>":
        <% else %>
            There are <%= nProductsFindSpec %> products that contain "<%= strFindSpec %>":
        <% end if %>
    <% end if %>
    <p>
    <table>
    <%
    set skuField = rsProductsFindSpec("sku")
    set dept_idField = rsProductsFindSpec("dept_id")
    set nameField = rsProductsFindSpec("name")
    set list_priceField = rsProductsFindSpec("list_price")
    %>
do while Not rsProductsFindSpec.EOF

<TD>
   <A HREF="<%= baseURL("product.asp") &
   mscsPage.URLShopperArgs("sku", skuField.value, "dept_i d", 
   dept_i dField.value,"epartner_i d=" epartner_i d) %>">
   <%= mscsPage.HTMLEncode(skuField.value) %>
   </A>
</TD>

<TD>
   <A HREF="<%= baseURL("product.asp") &
   mscsPage.URLShopperArgs("sku", skuField.value, "dept_i d", 
   dept_i dField.value,"epartner_i d=" epartner_i d) %>">
   <%= mscsPage.HTMLEncode(nameField.value)
   </A>
</TD>

<TD>
   <%= MSCSDataFunctions.Money(list_priceField.value) %>
</TD>

</TR>

rsProductsFindSpec.MoveNext
loop
</TABLE>
</TD>
</% end if %>
</% end if %>

<-- # include file="i_footer.asp" -->

</BODY>

</HTML>
global.asa

<OBJ ECT RUNAT=Server SCOPE=Application ID=MSCSSite
PROGID="Commerce.Dictionary" /></OBJ ECT>

<OBJ ECT RUNAT=Server SCOPE=Application ID=MSCSQueryMap
PROGID="Commerce.Dictionary" /></OBJ ECT>

<OBJ ECT RUNAT=Server SCOPE=Application ID=MSCSCache
PROGID="Commerce.Dictionary" /></OBJ ECT>

<OBJ ECT RUNAT=Server SCOPE=Application ID=MSCSMessageManager
PROGID="Commerce.MessageManager" /></OBJ ECT>

<OBJ ECT RUNAT=Server SCOPE=Application ID=MSCSDataFunctions
PROGID="Commerce.DataFunctions" /></OBJ ECT>

<OBJ ECT RUNAT=Server SCOPE=Application ID=MSCSShopperManager
PROGID="Commerce.StandardSManager" /></OBJ ECT>

<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Application_OnStart

vRoot = "FiveLakes"

Dim MSCSSite
Dim MSCSQueryMap
Dim MSCSMessageManager
Dim MSCSDataFunctions
Dim MSCSShopperManager

REM -- Read Store Dictionary
Set MSCSSite = ReadSiteDict(vroot)

REM -- Create a Query Map and add all queries:
set MSCSQueryMap = InitQueryMap()

REM -- Initialize Message Manager (for use in pipeline) and add all messages:
set MSCSMessageManager = InitMessageManager()

REM -- Initialize Shopper Manager for managing shopperId values
set MSCSShopperManager = InitShopperManager

REM -- Initialize Data Functions with locale:
set MSCSDataFunctions = InitDataFunctions

Set Application("MSCSSite") = MSCSSite
Set Application("MSCSQueryMap") = MSCSQueryMap
Set Application("MSCSMessageManager") = MSCSMessageManager
Set Application("MSCSDataFunctions") = MSCSDataFunctions
Set Application("MSCSShopperManager") = MSCSShopperManager
Application("MSCSSIDUrlKey") = "mscssid"

End Sub
Function InitShopperManager
    call MSCSShopperManager.InitManager(vRoot, "url")
    set InitShopperManager = MSCSShopperManager
End Function

Function InitDataFunctions
    MSCSDataFunctions.locale = 1033
    set InitDataFunctions = MSCSDataFunctions
End Function

Function ReadSiteDict(vRoot)
    REM -- Read Store Dictionary
    vRootDir = Server.MapPath("/" & vRoot)
    Call fileDocument.ReadDictionaryFromFile(vRootDir & 
"\config\site.csc", "IISProperties", MSCSSite)
    Set fileDocument = Nothing
    set ReadSiteDict = MSCSSite
End Function

Function InitQueryMap
    REM -- Create Query Map Dictionary
    Set MSCSQueryMap.related_products_upsell = AddQuery("Select sku, Upsell_sku, description from FiveLakes_promo_upsell where sku=:1")
    Set MSCSQueryMap.depts = AddQuery("SELECT dept_id, dept_name, dept_description FROM fivelakes_dept")
    Set MSCSQueryMap.dept_by_id = AddQuery("SELECT dept_id, dept_name, dept_description FROM fivelakes_dept WHERE dept_id = :1")
    Set MSCSQueryMap.products_by_dept = AddQuery("SELECT prod.sku, prod.name FROM fivelakes_product prod, fivelakes_dept_prod dept WHERE dept.sku = prod.sku and dept.dept_id = :1")
    Set MSCSQueryMap.price_promo_system = AddQuery("SELECT promo_name, date_start, date_end, shopper_all, shopper_column, shopper_op, shopper_value, cond_all, cond_column, cond_op, cond_value, cond_basis, award_all, award_column, award_op, award_value, award_max, disjoint_cond_award, disc_type, disc_value FROM fivelakes_promo_price WHERE active <> 0 ORDER BY promo_rank")
    Set MSCSQueryMap.related_products = AddQuery("SELECT prod.sku, prod.name FROM fivelakes_product prod, fivelakes_promo_cross prod WHERE promo_cross.skus = :1 and promo_cross.rel_skus = prod.sku")
    Set MSCSQueryMap.related_products_with_dept = AddQuery("SELECT prod.sku, prod.name, deptprod.dept_id FROM fivelakes_dept_prod deptprod WHERE promo_cross.skus = :1 and prod.sku = deptprod.sku and promo_cross.rel_skus = prod.sku")
Set _MSCSQueryMap.product_by_sku = AddQuery("SELECT pf.sku, pf.name, pf.description, pf.list_price, pf.sale_price, pf.sale_start, pf.sale_end, pf.image_file, pf.image_height, pf.author, pf.edition, pf.publisher, dept.dept_id, dept.dept_name FROM Fivelakes_product pf, Fivelakes_dept_prod deptprod, Fivelakes_dept dept WHERE pf.sku = :1 and pf.sku = deptprod.sku and dept.dept_id = deptprod.dept_id and dept.dept_id = :2")

Set _MSCSQueryMap.product_info = AddQuery("SELECT pf.sku, pf.name, pf.list_price, pf.sale_price, pf.sale_start, pf.sale_end, dept.dept_id FROM fivelakes_product pf, fivelakes_dept_prod deptprod, fivelakes_dept dept WHERE pf.sku = ? and pf.sku = deptprod.sku and dept.dept_id = deptprod.dept_id and dept.dept_id = ?")

Set _MSCSQueryMap.find_by_keyword = AddQuery("select p.sku, p.name, p.list_price, dp.dept_id from FiveLakes_product p, FiveLakes_dept_prod dp where p.name like '%:1%' and p.sku = dp.sku order by p.name")

Set _MSCSQueryMap.find_by_category = AddQuery("select p.sku, p.name, p.category, p.list_price, dp.dept_id from FiveLakes_product p, FiveLakes_dept_prod dp where p.category = ':1' and p.sku = dp.sku order by p.name")

Set _MSCSQueryMap.country = AddQuery("select country from FiveLakes_country")

Set _MSCSQueryMap.country_tax = AddQuery("select country, tax_rate from FiveLakes_country where country = :1 or code = :2")

Set _MSCSQueryMap.orderid = AddQuery("SELECT * FROM Fivelakes_receipt WHERE order_id = ':1'")

Set _MSCSQueryMap.email_search = AddQuery("SELECT * FROM Fivelakes_shopper WHERE email = ':1'")

Set _MSCSQueryMap.epartner = AddQuery("SELECT epartner_id, name, site_address FROM FiveLakes_ePartner where epartner_id = ':1'")

set InitQueryMap = _MSCSQueryMap
End Function

Function AddQuery(SQLCommand)
REM -- sets defaults that can be overridden per query on the page

REM -- ADO command types
adCmdText = 1
adCmdTable = 2
adCmdStoredProc = 4
adCmdUnknown = 8

REM -- ADO cursor types
adOpenForwardOnly = 0 # (Default)
adOpenKeyset = 1
adOpenDynamic = 2
adOpenStatic = 3

REM -- ADO lock types
adLockReadOnly = 1
adLockPessimistic = 2
adLockOptimistic = 3
adLockBatchOptimistic = 4

Set query = Server.CreateObject("Commerce.Dictionary")

query.SQLCommand = SQLCommand
query.Timeout = 0
query.CommandType = adCmdText
query.MaxRows = 0
query.CursorType = adOpenStatic
query.CursorSize = 0

Set AddQuery = query

End Function

Function InitMessageManager
    call MSCSMessageManager.AddLanguage("USA", 1033)
    MSCSMessageManager.defaultLanguage = "USA"
    call MSCSMessageManager.AddMessage("pur_out_of_stock", "At least one item is out of stock.")
    call MSCSMessageManager.AddMessage("pur_badsku", "Please note that one or more items were removed from your basket because the product is no longer sold.")
    call MSCSMessageManager.AddMessage("pur_badplacedprice", "Please note that prices of products in your basket have been updated.")
    call MSCSMessageManager.AddMessage("pur_noitems", "An order must have at least one item")
    call MSCSMessageManager.AddMessage("pur_badshipping", "Unable to complete order: cannot compute shipping cost.")
    call MSCSMessageManager.AddMessage("pur_badtax", "Unable to complete order: cannot compute tax.")
    call MSCSMessageManager.AddMessage("pur_badhandling", "Unable to complete order: cannot compute handling cost.")
    call MSCSMessageManager.AddMessage("pur_badverify", "Changes to the data require your review. Please review and resubmit.")
    call MSCSMessageManager.AddMessage("pur_badpayment", "There was a problem authorizing your credit. Please verify your payment information or use a different card.")
    call MSCSMessageManager.AddMessage("pur_badcc", "The credit-card number you provided is not valid. Please verify your payment information or use a different card.")
    call MSCSMessageManager.AddMessage("val_noshipmethod", "No Shipping Method was specified")
    call MSCSMessageManager.AddMessage("val_shiponeof", "Shipping Method Type must be one of: ")
    call MSCSMessageManager.AddMessage("val_noshipname", "No Ship-to Name was specified")
call MSCSMessageManager.AddMessage("val_noshi pstreet", "No Ship-to Street was specified")
call MSCSMessageManager.AddMessage("val_noshi pcity", "No Ship-to City was specified")
call MSCSMessageManager.AddMessage("val_noshi pstate", "No Ship-to State was specified")
call MSCSMessageManager.AddMessage("val_noshi pzip", "No Ship-to ZIP Code was specified")
call MSCSMessageManager.AddMessage("val_noshi pprovince", "No Ship-to Province was specified")
call MSCSMessageManager.AddMessage("val_noshi ppostal_code", "No Ship-to Postal Code was specified")
call MSCSMessageManager.AddMessage("val_noshi paddress1", "No Ship-to Address 1 was specified")
call MSCSMessageManager.AddMessage("val_noshi paddress2", "No Ship-to Address 2 was specified")
call MSCSMessageManager.AddMessage("val_noshi pcountry", "No Ship-to Country was specified")
call MSCSMessageManager.AddMessage("val_shi pcountrymust be", "The Ship-to Country must be United States.")
call MSCSMessageManager.AddMessage("val_nobillname", "No Bill-to Name was specified")
call MSCSMessageManager.AddMessage("val_nobillstreet", "No Bill-to Street was specified")
call MSCSMessageManager.AddMessage("val_nobillcity", "No Bill-to City was specified")
call MSCSMessageManager.AddMessage("val_nobillstate", "No Bill-to State was specified")
call MSCSMessageManager.AddMessage("val_nobillzip", "No Bill-to ZIP Code was specified")
call MSCSMessageManager.AddMessage("val_nobillprovince", "No Bill-to Province was specified")
call MSCSMessageManager.AddMessage("val_nobillpostal_code", "No Bill-to Postal Code was specified")
call MSCSMessageManager.AddMessage("val_nobilladdress1", "No Bill-to Address 1 was specified")
call MSCSMessageManager.AddMessage("val_nobilladdress2", "No Bill-to Address 2 was specified")
call MSCSMessageManager.AddMessage("val_nobillcountry", "No Bill-to Country was specified")
call MSCSMessageManager.AddMessage("val_noccname", "No Credit-card Name was specified")
call MSCSMessageManager.AddMessage("val_nocctype", "No Credit-card Type was specified")
call MSCSMessageManager.AddMessage("val_cctypeoneof", "Credit-card Type must be one of: ")
call MSCSMessageManager.AddMessage("val_noccnumber", "No Credit-card Number was specified")
call MSCSMessageManager.AddMessage("val_invalidccdate", "Credit-card expiration date must contain a valid month (1-12) and year (1997-2003)"")

set InitMessageManager = MSCSMessageManager
End Function
</SCRIPT>
<% REM Check for E-partner
epartner_id=0
epartner_id=request.QueryString("epartner_id")
if epartner_id=0 then
    epartner_id=1
else
    cmdTemp.CommandText = Replace(MSCSQueryMap.epartner.SQLCommand,":1", epartner_id)
    Set rsepartner = Server.CreateObject("ADODB.Recordset")
    rsepartner.Open cmdTemp, , adOpenForwardOnly, adLockReadOnly
    if rsepartner.EOF then
        epartner_id=1
    end if
end if
end if
%>
We are sorry, but we are unable to process your request:

We are sorry, but we are unable to process your request.

Please go back and correct the error and try again.

Please go back and correct the error and try again.
<% if IsEmpty(flag_navbar) then %>
<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" ALIGN="LEFT">
  <TD VALIGN="TOP">
    <TR> <TD ALIGN="RIGHT" VALIGN="TOP">
      <A HREF="<%= pageURL("about.asp") %>" %>%&gt; MG SRC="&lt;%= "/" & siteRoot %>/manager/MSCS_Images/navbar/btnabout.gif" W DTH="53" H E I G H T="41" BORDER="0" ALT="About" ALI GN="TOP" &gt;&lt;/A&gt;
        &lt;/TD&gt;&lt;TD W D T H="8" ROWSPAN="5"&nbsp; &lt;/TD&gt;&lt;/TR&gt;

    &lt;TR&gt; &lt;TD ALI GN="RI GHT" VALI GN="TOP"&gt;
      &lt;A HREF="<%= pageURL("default.asp") %>" %>%&gt; MG SRC="&lt;%= "/" & siteRoot %>/manager/MSCS_Images/navbar/btnmenu.gif" W DTH="53" H E I G H T="41" BORDER="0" ALT="Lobby" ALI GN="TOP" &gt;&lt;/A&gt;
        &lt;/TD&gt;&lt;/TR&gt;

    &lt;TR&gt; &lt;TD ALI GN="RI GHT" VALI GN="TOP"&gt;
      &lt;A HREF="<%= pageURL("find.asp") %>" %>%&gt; MG SRC="&lt;%= "/" & siteRoot %>/manager/MSCS_Images/navbar/btnfind.gif" W DTH="53" H E I G H T="41" BORDER="0" ALT="Find" ALI GN="TOP" &gt;&lt;/A&gt;
        &lt;/TD&gt;&lt;/TR&gt;

    &lt;TR&gt; &lt;TD ALI GN="RI GHT" VALI GN="TOP"&gt;
      &lt;A HREF="<%= pageURL("basket.asp") %>" %>%&gt; MG SRC="&lt;%= "/" & siteRoot %>/manager/MSCS_Images/navbar/btnbskt.gif" W D TH="53" H E I G H T="41" BORDER="0" ALT="Basket" ALI GN="TOP" &gt;&lt;/A&gt;
        &lt;/TD&gt;&lt;/TR&gt;

    &lt;TR&gt; &lt;TD ALI GN="RI GHT" VALI GN="TOP"&gt;
      &lt;A HREF="<%= pageURL("shipping.asp") %>" %>%&gt; MG SRC="&lt;%= "/" & siteRoot %>/manager/MSCS_Images/navbar/btnpay.gif" W D TH="53" H E I G H T="41" BORDER="0" ALT="Pay" ALI GN="TOP" &gt;&lt;/A&gt;
        &lt;/TD&gt;&lt;/TR&gt;

  &lt;/TD&gt;&lt;/TABLE&gt;
&lt;% end if %&gt;
i_mswallet.asp

<% 
REM -- For intranets not connected to the internet, override default
REM   download location here. For example:
' If LCase(CStr(Request("HTTP_UA_CPU"))) <> "alpha" Then
'     strMSWltEDwnldLoc  = "/" & siteRoot & 
"/manager/MSCS_Images/controls/MSWallet.cab"
'   Else
'     strMSWltEDwnldLoc  = "/" & siteRoot & 
"/manager/MSCS_Images/controls/MSWltAlp.cab"
'   End If
' strMSWltNavDwnldLoc = "/" & siteRoot & 
"/manager/MSCS_Images/controls"

REM -- Set wallet control accept credit card types.
strMSWltAcceptedTypes = 
"visa:clear;mastercard:clear;amex:clear;"

REM -- use_form set to 1 as a Request parameter will force the downlevel page
%>
<!--#INCLUDE FILE="i_selector.asp" -->
i_selector.asp

<% ' File Version 2,1,0,1378

' Variables set before including this file.

' One of the following two variables should be set before including this file:
  fMSWltAddressSelector   Set to "True" when the AddressSelector appears on the page.
  fMSWltPaymentSelector   Set to "True" when the PaymentSelector appears on the page.

' The following variables may be optionally set before including this file:
  strMSWltIEDwnldLoc      When not connected to the Internet, set to the location of the mswallet.cab, the wallet download package for IE.
  strMSWltNavDwnldLoc     When not connected to the Internet, set to the location of the HTM page containing download instructions (plginst.htm in the SDK).
  strMSWltDwnldVer        Allows overriding the downloaded version of Microsoft Wallet. Overriding the version number should not be necessary, so don't use this unless you understand what you're doing.
  fMSWltShowErrorDialogs  For debugging, set this to "True" to show error dialogs in GetValue calls.

' Variables set as a side effect of using including this file. You can use these in your ASP code:
  fMSWltActiveXBrowser supports ActiveX.
  fMSWltLiveConnectBrowser NOT support ActiveX and
    This covers Nav3 and probably Nav4.
  fMSWltUplevel Browser True if either of
    fMSWltActiveXBrowser or
    -- e.g., it's a control
  fMSWltLiveConnectBrowser --
Appendix A: Lab and Practice Solution Files

Server Side VBScript APIs

- MSWltIEAddrSelectorClassid
- MSWltIEPaySelectorClassid
  Returns the `<OBJECT>` CLASSID field. Always use as the CLASSID value. The returned classid is different on an Alpha NT machine.

- MSWltIECodebase()
  Returns the `<OBJECT>` CODEBASE field. Always use as the CODEBASE value.

- MSWltNavDwnldURL(strInstructionsFileName)
  Returns the `<EMBED>` PLUGINSPAGE field. Always use as the PLUGINSPAGE value. strInstructionsFileName specifies the name of the instructions file name. When using the default download location, call
  MSWltNavDwnldURL with 'plginst.htm'. When using an Intranet download location, call MSWltNavDwnldURL with the name of the plugin instructions file (e.g., plginst.htm in the SDK).

- MSWltLastChanceText(strDownlevelURL)
  Returns HTML text to place in your page. This text puts a link reading "Click here if you have problems with the Microsoft Wallet". When this link is clicked, the user navigates to the strDownlevelURL parameter, which in many cases will be the same page. This routine automatically appends "use_form=1" to strDownlevelURL to force the downlevel version of the page. We recommend setting the font to size 1.

- MSWltLoadDone(strDownlevelURL)
  Must be called in the uplevel browser case (fMSWltUplevelBrowser = True) on the `<BODY>` OnLoad field. Pass in the downlevel URL, as with MSWltLastChanceText. This is only used when the user refuses the Nav plugin on initial install or subsequent upgrade.

Client Side JavaScript APIs

- MSWltLoadDone(strDownlevelURL)
Use the server-side version or strDownlevel URL won't be set correctly.

MSWltPrepareForm(form, cParams, xlationArray)
Use this to update the fields in form with values from the AddressSelector and/or PaymentSelector. cParams is the count of the total number of parameters passed, including the cParams parameter. xlationArray is an optional set of parameters used to translate when formfield names to not match field names returned by the selectors; see the documentation for examples. Call this routine before posting the form. This routine will routine true if it succeeds, so it can be placed in a OnSubmit event handler, but this is not recommended, because if any JavaScript errors occur, the post will happen anyway erroneously. In IE3 when using multiple frames and JavaScript-based navigation between frames, using this routine causes subsequent JavaScript-based navigation to fail by navigating to the wrong location. We recommend not using this routine in those cases. This only applies in the most complex multi-frame/JavaScript-based navigation cases, like Adventure Works in CommerceServer v3.00.

Prepare miscellaneous variables.
fMSWltAddressSelector = CBool(fMSWltAddressSelector)
fMSWltPaymentSelector = CBool(fMSWltPaymentSelector)
fMSWltShowErrorDialogs = CBool(fMSWltShowErrorDialogs)

Browser Detection.
Set objBrowser = Server.CreateObject("MSWC.BrowserType")
strCPU = LCase(CStr(Request("HTTP_UA_CPU"))) ' CPU is necessary to differentiate between alpha, x86 and other CPUs on NT. ' only set for IE, Nav doesn't set.
If strCPU = "alpha" Then
fMSWltAlphaIE = true
Else
fMSWltAlphaIE = false
End If

Note that use_form = 1 forces a downlevel page.
If Request.QueryString("use_form") = 0 And objBrowser.JavaScript = "True" And (_
(fMSWltAddressSelector Or fMSWltPaymentSelector) Then

If objBrowser.ActiveXControls = "True" Then
  fMSWltActiveXBrowser = True
  fMSWltUplevelBrowser = True
ElseIf objBrowser.Browser = "Netscape" And _
  (objBrowser.Platform = "Win95" Or _
  objBrowser.Platform = "Win98" Or _
  ((objBrowser.Platform = "WinNT" Or _
    objBrowser.Platform = "Win32") And _
    ((Len(strCPU) = 0) Or (strCPU = "x86"))) ) _
And _
  ((CInt(objBrowser.majorver) > 3) Or _
  ((CInt(objBrowser.majorver) = 3) And _
  (objBrowser.beta = "False"))) _
  ) Then
    fMSWltLiveConnectBrowser = True
    fMSWltUplevelBrowser = True
Else
  fMSWltActiveXBrowser = False
  fMSWltLiveConnectBrowser = False
  fMSWltUplevelBrowser = False
End If
Else
  fMSWltActiveXBrowser = False
  fMSWltLiveConnectBrowser = False
  fMSWltUplevelBrowser = False
End If

' Examine to see if the download version or location has _
' been overridden. These should be overridden only when _
' the consumer is not connected to the Internet (i.e, intranet _
' scenario). _
' When not overridden, the default locations and version on _
' Microsoft.com will be used. _

' Download version. _
If IsEmpty(strMSWtDwnldVer) Then
  strMSWtDwnldVer = "2,1,0,1378"
Else
  strMSWtDwnldVer = CStr(strMSWtDwnldVer)
End If

' IE Wallet version location. _
If IsEmpty(strMSWtEDwnl dLoc) Then
  If fMSWtAlphaIE Then
    strMSWtEDwnl dLoc = "mswltalp.cab"
  Else
    strMSWtEDwnl dLoc = "mswallet.cab"
  End If
Else

Else
```vbnet
strMSWtIEDwnl dLoc = CStr(strMSWtIEDwnl dLoc)
End If

' Navigator Wallet version location.
If IsEmpty(strMSWtNavDwnl dLoc) Then
strMSWtNavDwnl dLoc = "http://www.microsoft.com/commerce/wallet/local/
Else
strMSWtNavDwnl dLoc = CStr(strMSWtNavDwnl dLoc)
' add trailing blank if missing
If Right(strMSWtNavDwnl dLoc, 1) <> "/" Then
    strMSWtNavDwnl dLoc = strMSWtNavDwnl dLoc & "/"
End If
End If

Function MSWtIEAddrSelectorClassid
If fMSWtAlphaIE Then
    MSWtIEAddrSelectorClassid = "clsid:B7FB4D5B-9FBE-11d0-8965-0000F822DEA9"
Else
    MSWtIEAddrSelectorClassid = "clsid:87D3CB63-BA2E-11cf-B9D6-00A0C9083362"
End If
End Function

Function MSWtIEPaySelectorClassid
If fMSWtAlphaIE Then
    MSWtIEPaySelectorClassid = "clsid:B7FB4D5C-9FBE-11d0-8965-0000F822DEA9"
Else
    MSWtIEPaySelectorClassid = "clsid:87D3CB66-BA2E-11cf-B9D6-00A0C9083362"
End If
End Function

Function MSWtIECodebase
    MSWtIECodebase = strMSWtIEDwnl dLoc & ">#Version=" & 
End Function

Function MSWtNavDwnl dURL(strInstructionsFileName)
    MSWtNavDwnl dURL = strMSWtNavDwnl dLoc & 
End Function

' Tack use_form=1 on to the end of a URL
Function MSWtDownlevelURL(strDownlevelURL)
    MSWtDownlevelURL = Trim(CStr(strDownlevelURL)) ' any spaces lurking?  get rid of them
    nQmarkLoc = InStr(MSWtDownlevelURL, "?")
    If nQmarkLoc > 0 Then
        MSWtDownlevelURL = MSWtDownlevelURL & "use_form=1"
    Else
        MSWtDownlevelURL = MSWtDownlevelURL & "use_form=1"
    End If
End Function
```
MSWltDownlevelURL = MSWltDownlevelURL & "&use_form=1"
  Else
    MSWltDownlevelURL = MSWltDownlevelURL & "?use_form=1"
  End If
End Function

Function MSWltLastChanceText(strDownlevelURL)
  If fMSWltUplevelBrowser And (fMSWltAddressSelector Or fMSWltPaymentSelector) Then
    MSWltLastChanceText = "<A HREF="" & 
      MSWltDownlevelURL(strDownlevelURL) & 
      "&epartner_id=" & 
      "" >Click here if you have problems with the Wallet</A> "
  End If
End Function

Function MSWltLoadDone(strDownlevelURL)
  ' Call JavaScript MSWltLoadDone routine with downlevel URL
  MSWltLoadDone = "MSWltLoadDone('" & 
  MSWltDownlevelURL(strDownlevelURL) & 
  "')"
End Function

<script language="JavaScript">
  <!--
  var fMSWltLoaded = false   <!-- Has onLoad initialization been done? -->

  <!-- If fMSWltAddressSelector Then -->
  var objAddrSelector   <!-- Address selector from both IE and Nav -->
  <!-- End If -->

  <!-- If fMSWltPaymentSelector Then -->
  var objPaySelector    <!-- Payment selector from both IE and Nav -->
  <!-- End If -->

  // JavaScript version.
  function MSWltLoadDone(strDownlevelURL)
  {
    <!-- If fMSWltLiveConnectBrowser Then -->
    <!-- Is the plugin around? -->

    if (<!-- If fMSWltAddressSelector Then -->
        (document.addrSelector == null)
<% If fMSWltPaymentSelector Then %>
||
<% End If %>
<% End If %>
<% If fMSWltPaymentSelector Then %>
(document.paySelector == null)
<% End If %>
}

if (confirm("Click OK to install Microsoft Wallet Plugins."))
<% ' open instructions page in a new window %>
window.open("<%= MSWltNavDwnldURL("plginst.htm") %>
else
    location = strDownlevelURL
    return
}

<% ' Take care of naming differences between objects and plugins. %>
<% If fMSWltAddressSelector Then %>
    objAddrSelector = document.addrSelector
    fVersionOK = objAddrSelector.VersionCheck()
<% End If %>

<% If fMSWltAddressSelector And fMSWltPaymentSelector Then %>
    if (fVersionOK)
    {
        objPaySelector = document.paySelector
        fVersionOK = objPaySelector.VersionCheck()
    }
<% ElseIf fMSWltPaymentSelector Then %>
    objPaySelector = document.paySelector
    fVersionOK = objPaySelector.VersionCheck()
<% End If %>

<% ' Check plugin version. Version requested set on <embed> tag. %>
<% ' This version is checked against the version resource in the DLL. %>
    if (!fVersionOK)
    {
        if (confirm("Your Microsoft Wallet Plugins are out of date and you need to upgrade.\nClick OK to view upgrade directions."))
        <%' open instructions page in a new window %>
        window.open("<%= MSWltNavDwnldURL("plginst.htm") %>
    else
        location = strDownlevelURL
        return
    }

fMSWltLoaded = true
<% End If %>
}

function MSWltCheckLoaded() {
if (!fMSWltLoaded) {
  <% If fMSWltActiveXBrowser Then %>
  if (
    <% If fMSWltAddressSelector Then %>
      (!(!addrSelector))
    <% End If %>
    <% End If %>
    <% If fMSWltPaymentSelector Then %>
      (!(!paySelector))
    <% End If %>
    <% End If %>
  )
  
  <% If fMSWltAddressSelector Then %>
  objAddrSelector = addrSelector
  <% End If %>
  <% If fMSWltPaymentSelector Then %>
  objPaySelector = paySelector
  <% End If %>
  fMSWltLoaded = true
%
  else
    <% End If %>

    <% ' Navigator JavaScript does not take kindly to pushing buttons before the page is done %>
    <% ' loading; hence the warning that a reload and wait may be necessary. %>
    alert("Page not done loading yet. Try again when it's loaded. Refresh the page if you're having difficulty (then wait for it to load.")
    return fMSWltLoaded
  }
%

function doNothing() { <% ' Do nothing, supports object creation with no contents. %} }

function MSWltPrepareForm(form cParams, slationArray) {
  if (!MSWltCheckLoaded())
    return false

  <% If fMSWltPaymentSelector Then %>
  PI = objPaySelector.GetValues()           <% ' get payment information (PI) %>
  <% End If %>
}
errorStatus = objPaySelector.GetLastError()  <% ' did an error occur? %>
    if (errorStatus < 0)
    {
        if (errorStatus != (-2147220991) &&
            errorStatus != (-2147220990))  <% ' HRESULT 0x80040201,
            WALLET_E_CANCEL, and HRESULT 0x80040202, WALLET_E_HANDLEDERROR %>
            alert("Payment selection failed due to an unknown problem")
            return false
    }
}% End If %>

<% If fMSWltAddressSelector Then %>
    shipTo = objAddrSelector.GetValues()  <% ' get ship to address information %>
    errorStatus = objAddrSelector.GetLastError()  <% ' did an error occur? %>
    if (errorStatus < 0)
    {
        if (errorStatus != (-2147220991) &&
            errorStatus != (-2147220990))  <% ' HRESULT 0x80040201,
            WALLET_E_CANCEL, and HRESULT 0x80040202, WALLET_E_HANDLEDERROR %>
            alert("Address selection failed due to an unknown problem")
            return false
    }
}% End If %>

elements = form.elements

<% ' Build xlation table from xlationArray %>
xlate = new doNothing()
for (i = 2; i < cParams; i += 2)
{
    value = MSWltPrepareForm.arguments[i+1]
    if (value.length > 0)
        xlate[MSWltPrepareForm.arguments[i]] = value
}

<% If fMSWltPaymentSelector Or fMSWltAddressSelector %>
for (i = 0; i < elements.length; i++)
{
    if (form.elements[i].name.length > 0)
    {
        xlateValue = xlate[form.elements[i].name]
        if (xlateValue)
            name = xlateValue
        else
            name = form.elements[i].name
    }
<% If fMSWltPaymentSelector Then %>
  <‘ Have to make the string at least
  1 long to get around Nav issue %>
  value = 'a' +
  objPaySelector.GetValue(PI, name, <‘ = -
  CInt(fMSWltShowErrorDialogs) %>)
  if (value.length > 1)
    elements[i].value =
    value.substring(1)
  <‘ End If %>
<% End If %>

<% If fMSWltAddressSelector Then %>
  <‘ Have to make the string at least
  1 long to get around Nav issue %>
  value = 'a' +
  objAddrSelector.GetValue(shipTo, name, <‘ = -
  CInt(fMSWltShowErrorDialogs) %>)
  if (value.length > 1)
    elements[i].value =
    value.substring(1)
  <‘ End If %>
<% End If %>

  return true
</SCRIPT>
<% End If %>
<%}
function this_page(nowat)
    this_page = (Right(Request.ServerVariables("SCRIPT_NAME"),len(nowat)) = nowat)
end function

REM -- ADO command types
adCmdText = 1
adCmdTable = 2
adCmdStoredProc = 4
adCmdUnknown = 8

REM -- ADO cursor types
adOpenForwardOnly = 0 ' # (Default)
adOpenKeyset = 1
adOpenDynamic = 2
adOpenStatic = 3

REM -- ADO lock types
adLockReadOnly = 1
adLockPessimistic = 2
adLockOptimistic = 3
adLockBatchOptimistic = 4

REM -- Used to check ADO Supports for Oracle
adApproxPosition = 16384

REM -- If store is not open then redirect to closed URL
if MSCSSite.Status <> "Open" then
    response.redirect(MSCSSite.CloseRedirectURL)
end if

REM -- mscs = created on the page; MSCS = created in global.asa

REM -- Manually create shopper id
mscsShopperID = mscsPage.GetShopperID

REM - Handle shopper
if IsNull(mscsShopperID) then
    mscsShopperID = mscsShopperManager.CreateShopperID()
    mscsPage.PutShopperID(mscsShopperID)
end if

REM *****************************************************
REM -- functions for faster page links

i_shop.asp
function pageURL(pageName)
    pageURL = rootURL & pageName & "?" & emptyArgs
end function

function pageSURL(pageName)
    pageSURL = rootSURL & pageName & "?" & emptyArgs
end function

function baseURL(pageName)
    REM -- you must put on your own shopperArgs
    baseURL = rootURL & pageName & "?"
end function

function baseSURL(pageName)
    REM -- you must put on your own shopperArgs
    baseSURL = rootSURL & pageName & "?"
end function

displayName = MSCSSite.DisplayName
siteRoot = mcsPage.SiteRoot()
rootURL = mcsPage.URLPrefix() & "/" & siteRoot & "/
rootSURL = mcsPage.SURLPrefix() & "/" & siteRoot & "/
emptyArgs = mcsPage.URLShopperArgs()

REM *******************************************************

REM -- Create ADO Connection and Command Objects
Set MSCS = Server.CreateObject("ADODB.Connection")
MSCS.Open MSCSSite.DefaultConnectionString
Set cmdTemp = Server.CreateObject("ADODB.Command")
cmdTemp.CommandType = adCmdText
Set cmdTemp.ActiveConnection = MSCS
%>
<%
function UtilGetOrderFormStorage()
    Set orderFormStorage = Server.CreateObject("Commerce.DBStorage")
    Call orderFormStorage.InitStorage(MSCSSite.DefaultConnectionString,
        "fivelakes_basket", "shopper_id", "Commerce.OrderForm",
        "marshalled_basket", "date_changed")
    Set UtilGetOrderFormStorage = orderFormStorage
end function

function UtilGetOrderForm(byRef orderFormStorage, byRef created)
    created = 0
    On Error Resume Next
    Set orderForm = orderFormStorage.GetData(null, mscsShopperID)
    On Error Goto 0
    If IsEmpty(orderForm) Then
        set orderForm = Server.CreateObject("Commerce.OrderForm")
        orderForm.shopper_id = mscsShopperID
        created = 1
    End If
    set UtilGetOrderForm = orderForm
end function

function UtilPutOrderForm(byRef orderFormStorage, byRef orderForm, byRef created)
    If created = 0 Then
        Call orderFormStorage.CommitData(NULL, orderForm)
    Else
        Call orderFormStorage.InsertData(NULL, orderForm)
    End If
end function

function UtilGetReceiptStorage()
REM Create a storage object for receipts
    Set receiptStorage = Server.CreateObject("Commerce.DBStorage")
    Call receiptStorage.InitStorage(MSCSSite.DefaultConnectionString,
        "fivelakes_receipt", "order_id", "Commerce.OrderForm",
        "marshalled_receipt", "date_entered")
    receiptStorage.Mapping.Value("_total_total") = "total"
    Set UtilGetReceiptStorage = receiptStorage
end function

function UtilGetReceiptStorage()
REM Create a storage object for receipts
    Set receiptStorage = Server.CreateObject("Commerce.DBStorage")
    Call receiptStorage.InitStorage(MSCSSite.DefaultConnectionString,
        "fivelakes_receipt", "order_id", "Commerce.OrderForm",
        "marshalled_receipt", "date_entered")
    receiptStorage.Mapping.Value("_total_total") = "total"
    Set UtilGetReceiptStorage = receiptStorage
end function

end function
function UtilGetPipeContext()
    Set pipeContext = Server.CreateObject("Commerce.Dictionary")
    Set pipeContext("MessageManager") = MSCSMessageManager
    Set pipeContext("DataFunctions") = MSCSDataFunctions
    Set pipeContext("QueryMap") = MSCSQueryMap
    Set pipeContext("ConnectionStringMap") = MSCSSite.ConnectionStringMap
    pipeContext("SiteName") = displayName
    pipeContext("DefaultConnectionString") = MSCSSite.DefaultConnectionString
    pipeContext("Language") = "USA"
    Set UtilGetPipeContext = pipeContext
end function

function UtilRunPipe(file, orderForm, pipeContext)
    Set pipeline = Server.CreateObject("Commerce.MtsPipeline")
    Call pipeline.LoadPipe(Request.ServerVariables("APPL_PHYSICAL_PATH") & "config\" & file)
    REM Call pipeline.SetLogFile(Request.ServerVariables("APPL_PHYSICAL_PATH") & "config\pipeline.log")
    errorLevel = pipeline.Execute(1, orderForm, pipeContext, 0)
    UtilRunPipe = errorLevel
end function

function UtilRunTxPipe(file, orderForm, pipeContext)
    Set pipeline = Server.CreateObject("Commerce.MtsTxPipeline")
    Call pipeline.LoadPipe(Request.ServerVariables("APPL_PHYSICAL_PATH") & "config\txpipeline.log")
    REM Call pipeline.SetLogFile(Request.ServerVariables("APPL_PHYSICAL_PATH") & "config\txpipeline.log")
    errorLevel = pipeline.Execute(1, orderForm, pipeContext, 0)
    UtilRunTxPipe = errorLevel
end function

function UtilRunPlan()
    REM Create a storage object for the order forms
Set mscsOrderFormStorage = UtilGetOrderFormStorage()

REM Get the order form
Set mscsOrderForm = UtilGetOrderForm(mscsOrderFormStorage, created)

REM Get the basic pipe context
Set mscsPipeContext = UtilGetPipeContext()

REM Create and run the pipe
errorLevel = UtilRunPipe("plan.pcf", mscsOrderForm, mscsPipeContext)

REM Save the order form in case running the pipe made changes to the order form
if created then
    Call mscsOrderFormStorage.InsertData(null, mscsOrderForm)
else
    Call mscsOrderFormStorage.CommitData(null, mscsOrderForm)
end if

Set UtilRunPlan = mscsOrderForm
end function
%>
order.asp

```vbscript
<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<!--# INCLUDE FILE="include/Manager.asp" -->

<% REM header: %>
<% pageTitle = "Order Manager" %>
<HTML>
<HEAD>
<TITLE> <%= pageTitle %> </TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
<!--#INCLUDE FILE="include/mgmt_define.asp" -->
</HEAD>
</HTML>

<BODY TOPMARGIN="8" LEFTMARGIN="8" BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#FF0000" ALINK="#FF0000" VLINK="#FF0000">
<!--#INCLUDE FILE="include/mgmt_header.asp" -->

<% REM body: %>
<UL>
  <LI> <A HREF="order_ePartner.asp"> Orders by ePartner </A>
  <LI> <A HREF="order_list.asp"> All Orders </A>
  <LI> <A HREF="order_day.asp"> Orders by Month </A>
  <LI> <A HREF="order_month.asp"> Orders by Year </A>
  <LI> <A HREF="order_product.asp"> Orders by Product </A>
</UL>

<% REM footer: %>
<!--#INCLUDE FILE="include/mgmt_footer.asp" -->
```
<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>
<!-# INCLUDE FILE="include/Manager.asp" -->
<% pageTitle="Orders By ePartner" %>

<HTML>
<HEAD>
  <TITLE><%= pageTitle %></TITLE>
  <META NAME="Generator" CONTENT="Microsoft Visual Studio 6.0">
  <!-# INCLUDE FILE="include/mgmt_define.asp" -->
</HEAD>

<BODY>
  <!-# INCLUDE FILE="include/mgmt_header.asp" -->
  <%- Set mscsReceiptStorage = Server.CreateObject("Commerce.DBSStorage") On Error Resume Next Call mscsReceiptStorage.InitStorage(mscsManagerSite.DefaultConnectionString, "Fivelakes_receipt", "order_id", "Commerce.OrderForm", "marshalled_receipt", "date_entered") %>
  if Err.Number <> 0 then
    Set errorList = Server.CreateObject("Commerce.SimpleList") errorList.Add "The table could not be found or the database connection failed."
  %>
  <!-# INCLUDE FILE="include/error.asp" -->
  <!-# INCLUDE FILE="include/mgmt_footer.asp" -->
  <%- function ShowRow()
    on error resume next
    set receipt = mscsReceiptStorage.GetData(null, CStr(rsList("order_id"))")
    on error goto 0
    if IsEmpty(receipt) then
      nitems = 0
    else
      set items = receipt.items
      nitems = items.Count
    end if %>

---

**order_ePartner.asp**

```vbscript
<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>
<!-# INCLUDE FILE="include/Manager.asp" -->
<% pageTitle="Orders By ePartner" %>

<HTML>
<HEAD>
  <TITLE><%= pageTitle %></TITLE>
  <META NAME="Generator" CONTENT="Microsoft Visual Studio 6.0">
  <!-# INCLUDE FILE="include/mgmt_define.asp" -->
</HEAD>

<BODY>
  <!-# INCLUDE FILE="include/mgmt_header.asp" -->
  <%- Set mscsReceiptStorage = Server.CreateObject("Commerce.DBSStorage") On Error Resume Next Call mscsReceiptStorage.InitStorage(mscsManagerSite.DefaultConnectionString, "Fivelakes_receipt", "order_id", "Commerce.OrderForm", "marshalled_receipt", "date_entered") %>
  if Err.Number <> 0 then
    Set errorList = Server.CreateObject("Commerce.SimpleList") errorList.Add "The table could not be found or the database connection failed."
  %>
  <!-# INCLUDE FILE="include/error.asp" -->
  <!-# INCLUDE FILE="include/mgmt_footer.asp" -->
  <%- function ShowRow()
    on error resume next
    set receipt = mscsReceiptStorage.GetData(null, CStr(rsList("order_id"))")
    on error goto 0
    if IsEmpty(receipt) then
      nitems = 0
    else
      set items = receipt.items
      nitems = items.Count
    end if %>
```
Appendix A: Lab and Practice Solution Files

```html
<% = RowCount %>

<% = rsList("epartner_id").value %>

<% = rsList("order_id").value %>

<% = MSCSDataFunctions.Date(rsList("date_entered").value) %>

<% = nitems %>

<% = MSCSDataFunctions.Money(rsList("total").value) %>

<%  end function %>

listElemTemplate = "order_view.asp"
listColumns = "<TH ALIGN="LEFT"> # </TH> & vbCr & _
"<TH ALIGN="LEFT"> ePartner ID </TH> & vbCr & _
"<TH ALIGN="LEFT"> Order ID </TH> & vbCr & _
"<TH ALIGN="LEFT"> Date </TH> & vbCr & _
"<TH ALIGN="LEFT"> # Items </TH> & vbCr & _
"<TH ALIGN="LEFT"> Total </TH> & vbCr & _
listNoRows = "<i>No orders in table</i>" %>

<% cmdTemp.CommandText = "SELECT order_id,epartner_id,shopper_id,date_entered,total,status FROM Fivelakes_receipt ORDER BY epartner_id" %>

<--# INCLUDE FILE="include/list.asp" -->
<--# INCLUDE FILE="include/mgmt_footer.asp" -->

</BODY>
</HTML>
```
order_status.asp

<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<% Response.buffer = true %>
<!--#INCLUDE FILE="i_shop.asp" -->
<!--#INCLUDE FILE="i_ePartner.asp" -->
<!--#INCLUDE FILE="i_header.asp" -->

<HTML>

<HEAD>
  <!--#INCLUDE FILE="i_header.asp" -->
  <META NAME="Generator" CONTENT="Microsoft Visual Studio 6.0">
</HEAD>

<BODY>
  <!--#INCLUDE FILE="i_header.asp" -->
  <%cmdTemp.CommandText = Replace(MSCSQueryMap.orderid.SQLCommand,":1", Request("order_track_id")) %>
  Set rsorderid = Server.CreateObject("ADODB.Recordset")
  rsorderid.Open cmdTemp, , adOpenForwardOnly, adLockReadOnly
  if not rsorderid.EOF then
    if (now() - rsorderid("date_entered"))>2 then
      ship_status="Yes"
    else
      ship_status="No"
    end if
  end if
%

<% strurl = "receipt.asp?order_id=" & Request("order_track_id") & "&ship_status=" & ship_status %>
  Response.Redirect strurl
%
<br>
</BODY>

</HTML>
ordertrack.asp

<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<!--# INCLUDE FILE="i_shop.asp" -->
<!--# INCLUDE FILE="i_ePartner.asp" -->
<!--# INCLUDE FILE="i_header.asp" -->
<HTML>

<HEAD>
  <META NAME="Generator" CONTENT="Microsoft Visual Studio 6.0">
</HEAD>

<BODY>
<form name="order_status" method="post" action="order_status.asp">
<table width="100%">
  <tr>
    <td width="20%" >
      Order Number:
    </td>
    <td>
      <input type="text" name="order_track_id" size="10" maxlength="100">
    </td>
  </tr>
  <tr>
    <td colspan="2">
    </td>
  </tr>
  <tr>
    <td width="20%" >
      <input type="submit" value="Go">
    </td>
    <td>
      <input type="reset" value="Reset">
    </td>
  </tr>
</table>
</form>
<p>&nbsp;</p>
</BODY>
</HTML>
payment.asp

```vbscript
<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<% Response.ExpiresAbsolute=DateAdd("yyyy", -10, Date) %>

<%@ Include File="i_shop.asp" -->
<%@ Include File="i_ePartner.asp" -->
<%@ Include File="i_util.asp" -->

<%
REM Run the basic plan
Set mscsOrderForm = UtilRunPlan()

set mscsOrderItems = mscsOrderForm.Items
nOrderItems = mscsOrderItems.Count

set mscsBasketErrors = mscsOrderForm._Basket_Errors
nBasketErrors = mscsBasketErrors.Count

set mscsPurchaseErrors = mscsOrderForm._Purchase_Errors
nPPurchaseErrors = mscsPurchaseErrors.Count

strDownlevelURL = pageSURL("payment.asp") & "&epartner_id=" & epartner_id %>

<HTML>

<HEAD>
<TITLE><%= displayName %>: Payment</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
<% fMSWltPaymentSelector = True %>
<%@ Include File="i_mswallet.asp" -->
<% if fMSWltUplevelBrowser and CBool(nOrderItems > 0) then %>
<SCRIPT LANGUAGE="Javascript">
  function submitPayinfo()
  {
    if (MSWltPrepareForm(document.payinfo, 2)) {
      document.payinfo.submit();
    }
  }
</SCRIPT>
<% end if %>
</HEAD>

<BODY>
```
Final Purchase Approval

If nOrderItems = 0 then

    <strong>Your basket is empty.</strong>

else

    If nBasketErrors > 0 then

        <table width="500">
            <tr><td><font color="#FF0000"><strong>mscsBasketErrors(iError)</strong></font></td></tr>
        </table>
    </if>

    If nPurchaseErrors > 0 then

        <table width="500">
            <tr><td><font color="#FF0000"><strong>mscsPurchaseErrors(iError)</strong></font></td></tr>
        </table>
    </if>

    Please go back to the basket page, correct the error, and try again.

else

    If Request("error").Count <> 0 then

        <table width="500">
            <tr><td><font color="#FF0000"><strong>Your purchase could not be completed.</strong></font></td></tr>
        </table>

    </if>
Please make sure you have provided all the requested billing and payment information.

Please enter your payment information and press the "Purchase" button below.

<table>
<thead>
<tr>
<th>Subtotal:</th>
<th>$\text{MSCSDataFunctions.Money(mscsOrderForm._total_total)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shipping:</td>
<td>$\text{MSCSDataFunctions.Money(mscsOrderForm._shipping_total)}$</td>
</tr>
<tr>
<td>Handling:</td>
<td>$\text{MSCSDataFunctions.Money(mscsOrderForm._handling_total)}$</td>
</tr>
</tbody>
</table>
### Credit Card Information

<table>
<thead>
<tr>
<th><strong>Name on card:</strong></th>
<th><strong>Card Number:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;input type=&quot;text&quot; name=&quot;cc_name&quot; size=&quot;70,1&quot;&gt;</td>
<td>&lt;input type=&quot;text&quot; name=&quot;_cc_number&quot; size=&quot;70,1&quot;&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Type:</strong></th>
<th><strong>Expiration Date:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;select name=&quot;cc_type&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;option value=&quot;Visa&quot;&gt; VISA &lt;/option&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;option value=&quot;MasterCard&quot;&gt; MasterCard &lt;/option&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;option value=&quot;AMEX&quot;&gt; American Express &lt;/option&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;select name=&quot;_cc_expmonth&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;%= mscsPage.Option(1, iMonth) %&gt; Jan</td>
<td></td>
</tr>
<tr>
<td>&lt;%= mscsPage.Option(2, iMonth) %&gt; Feb</td>
<td></td>
</tr>
<tr>
<td>&lt;%= mscsPage.Option(3, iMonth) %&gt; Mar</td>
<td></td>
</tr>
<tr>
<td>&lt;%= mscsPage.Option(4, iMonth) %&gt; Apr</td>
<td></td>
</tr>
<tr>
<td>&lt;%= mscsPage.Option(5, iMonth) %&gt; May</td>
<td></td>
</tr>
</tbody>
</table>
<\%= mscsPage.Option(6, iMonth) %\> Jun
<\%= mscsPage.Option(7, iMonth) %\> Jul
<\%= mscsPage.Option(8, iMonth) %\> Aug
<\%= mscsPage.Option(9, iMonth) %\> Sep
<\%= mscsPage.Option(10, iMonth) %\> Oct
<\%= mscsPage.Option(11, iMonth) %\> Nov
<\%= mscsPage.Option(12, iMonth) %\> Dec
</SELECT>
</TD>
</TD>

<\%= iYear = Year(Date) %\>
<\%= mscsPage.Option(1999, iYear) %\> 1999
<\%= mscsPage.Option(2000, iYear) %\> 2000
<\%= mscsPage.Option(2002, iYear) %\> 2002
<\%= mscsPage.Option(2003, iYear) %\> 2003
</SELECT>
</TD>
</TR>

<TR>
<TD>

&nbsp;
</TD>
</TR>

<\%= REM bill-to: %\>
<TR>
<TD><TD>
<TD ALIGN="CENTER" COLSPAN="3">%
<FONT SIZE="1">Billing Address</FONT>%<STRONG>Billing Address</STRONG></FONT>%
</TD>
</TR>

<TR>
<TD ALIGN="RIGHT">%<B>Name:</B>%
</TD>
<TD>COLSPAN="3">
<INPUT TYPE="text" NAME="bill_to_name" SIZE="54,1" VALUE=""><%= mscsPage.HTMLEncode(mscsOrderForm.bill_to_name) %">%>
</TD>
</TR>

<TR>
<TD ALIGN="RIGHT">%<B>Street:</B>%
</TD>
</TR>
| City: |  |
| State: |  |
| Country: |  |
<select name="bill_to_country" size="1">
    <%While not rscountry.EOF %>
        <%= mscsPage.Option(rscountry("country").value, mscsOrderForm.bill_to_country) %>
        <%= rscountry("country").value %>
    <% rscountry.MoveNext Wend %>
    </select>
</td>
</TR>

<TR>
    <TD ALIGN="RIGHT">
        <B>Phone:</B>
    </TD>
    <TD COLSPAN="3">
        <INPUT TYPE="text" NAME="bill_to_phone" SIZE="54">
    </TD>
</TR>

<TR>
    <TD>&nbsp;</TD>
</TR>

<TR>
    <TD ALIGN="RIGHT"></TD>
    <TD COLSPAN="4">
        <INPUT TYPE="Image" VALUE="Purchase" SRC="<%= "/&amp;siteRoot manager/MSCS_Images/buttons/btnpurchase1.gif" W DTH="116" H EIGHT="25" BORDER="0" ALT="Purchase">
    </TD>
</TR>
</TABLE>
</FORM>
<% end if ' uplevel %>
<% end if %>
<% end if %>

<--- # INCLUDE FILE="i_footer.asp" -->

</BODY>
</HTML>
product.asp

```vbscript
<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<!--# INCLUDE FILE="i_shop.asp" -->
<!--# INCLUDE FILE="i_ePartner.asp" -->
<%  
    sku = mcsPage.RequestString("sku")
    quoted_sku = "'" & Replace(sku,"'","''") & "'"    REM - add quotes

REM -- retrieve product:
    sqlText = MSCSQueryMap.product_by_sku.SQLCommand
    sqlText = Replace(sqlText, ":1", quoted_sku)
    sqlText = Replace(sqlText, ":2", Request("dept_id"))
    cmdTemp.CommandText = sqlText
    Set rsProduct = Server.CreateObject("ADODB.Recordset")
    rsProduct.Open cmdTemp, , adOpenForwardOnly, adLockReadOnly
    if rsProduct.EOF then
        product_exists = false
    else
        product_exists = true
    
    REM -- get fields from recordset
    author = rsProduct("author").value
    publisher = rsProduct("publisher").value
    edition = rsProduct("edition").value
    name = rsProduct("name").value
    description = rsProduct("description").value
    dept_name = rsProduct("dept_name").value
    list_price = rsProduct("list_price").value
    sale_price = rsProduct("sale_price").value
    sale_start = rsProduct("sale_start").value
    sale_end = rsProduct("sale_end").value
    image_file = rsProduct("image_file").value
    image_width = rsProduct("image_width").value
    image_height = rsProduct("image_height").value
    rsProduct.Close

REM -- Log department and product for UA

REM -- determine if product is on sale:
    today = Date
```
on_sale = DateDiff("d", today, sale_start) <= 0 and
     DateDiff("d", today, sale_end) > 0
end if

<HTML>

<HEAD>
    <TITLE><%= displayName %>: Product<% if product_exists
    then %>: '<%= mscsPage.HTMLEncode(name) %>'<% end if
    %></TITLE>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html;
    charset=ISO-8859-1">
</HEAD>

<BOky
    BGColor="#FFFFFF"
    TEXT= "#000000"
    LINK= "#FF0000"
    VLINK= "#FF0000"
    ALINK= "#FF0000"
>

<%-- # INCLUDE FILE="i_header.asp" -->

<% if not product_exists then %>
    <p>The product you requested is currently not available.
<% else %>

<FORM METHOD=POST ACTION="<%= pageSURL("xt_orderform_additem.asp") & 
&epartner_id=" 
&epartner_id %>">
    <INPUT TYPE="HIDDEN" NAME="sku" VALUE="<%= mscsPage.HTMLEncode(sku) %>">
    <INPUT TYPE="HIDDEN" NAME="dept_id" VALUE="<%= Request("dept_id") %>">
    <TABLE
        BORDER= "0"
        CELLPADDING="2"
        CELLSPACING="2"
    >
        <TR>
            <TD>
                <P><FONT SIZE="6"><B><%= mscsPage.HTMLEncode(name) %></B></FONT>
                <% if on_sale then %>

<% if on_sale then %>
ON SALE! 

Regular Price: 

Author 

Publisher 

Edition 

Add to Basket
<% REM get related products (if any):
    cmdTemp.CommandText = Replace("SELECT prod.sku, prod.name, deptprod.dept_id FROM fivelakes_promo_cross promo_cross, fivelakes_product prod, fivelakes_dept_prod deptprod WHERE promo_cross.sku = :1 and prod.sku = deptprod.sku and promo_cross.rel_sku = prod.sku", ":1", quoted_sku)
    Set rsRelated = Server.CreateObject("ADODB.Recordset")
    rsRelated.Open cmdTemp, , adOpenForwardOnly, adLockReadOnly
    REM display up to 5 related products:
    if Not rsRelated.EOF then
    %>
        &lt;BR&gt;
        &lt;B&gt;See Also&lt;/B&gt;
        &lt;%
        nRelated = 0
        set skuField = rsRelated("sku")
        set nameField = rsRelated("name")
        set dept_idField = rsRelated("dept_id")
        do while Not (rsRelated.EOF Or nRelated &gt;= 5)
        %&gt;&lt;BR&gt;
            &lt;A HREF="% = baseURL(\"product.asp\") &amp; mscsPage.URLShopperArgs(\"sku\", skuField.value, \"dept_id\", dept_idField.value, \"epartner_id\", epartner_id) %&gt; &lt;% = mscsPage.HTMLEncode(nameField.value) %&gt; &lt;/A&gt;
            nRelated = nRelated + 1
        rsRelated.MoveNext
        loop %
    &lt;% end if %&gt;
    &lt;p&gt;
    &lt;!--# INCLUDE FILE="i_footer.asp" --&gt;
    &lt;/BODY&gt;
    &lt;/HTML&gt;
<%@ LANGUAGE=vbscript enableSessionState=false LCID=1033 %>

<-- # N克莱 INCLUDE FILE="i_shop.asp" -->
<-- # N克莱 INCLUDE FILE="i_ePartner.asp" -->
<%  
sku = mcsPage.RequestString("sku")
quoted_sku = "'" & Replace(sku,"'","'"') & "'"    REM -
add quotes
index=Request.QueryString("index")
department_id=mcsPage.RequestNumber("department_id")

REM -- retrieve product:
sqlText = MSCSQueryMap.product_by_sku.SQLCommand
sqlText = Replace(sqlText, ":1", quoted_sku)
sqlText = Replace(sqlText, ":2", Request("department_id"))
cmdTemp.CommandText = sqlText
Set rsProduct = Server.CreateObject("ADODB.Recordset")
rsProduct.Open cmdTemp, , adOpenForwardOnly, adLockReadOnly

if rsProduct.EOF then
    product_exists = False
else
    product_exists = True

    REM -- get fields from recordset
    author = rsProduct("author").value
    publisher = rsProduct("publisher").value
    edition = rsProduct("edition").value
    name = rsProduct("name").value
    description = rsProduct("description").value
    department_name = rsProduct("department_name").value
    list_price = rsProduct("list_price").value
    sale_price = rsProduct("sale_price").value
    sale_start = rsProduct("sale_start").value
    sale_end = rsProduct("sale_end").value
    image_file = rsProduct("image_file").value
    image_width = rsProduct("image_width").value
    image_height = rsProduct("image_height").value

    rsProduct.Close

    REM -- Log department and product for UA
REM -- determine if product is on sale:
today = Date 
on_sale = DateDiff("d", today, sale_start) <= 0 and 
DateDiff("d", today, sale_end) > 0

end if
%

<html>
<head>
<title><%= displayName %>: Product<% if product_exists then %>: '<%= mscsPage.HTMLEncode(name) %>'<% end if %></title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head>
<body bgcolor="#FFFFFF" text="#000000" link="#FF0000" vlink="#FF0000" alink="#FF0000">
<!--#include file="i_header.asp" -->
<% if not product_exists then %>
<p>The product you requested is currently not available.</p>
<% else %>
<form method=POST action="%= pageSURL("xt_orderform_edititem.asp")& &epartner_id=" &epartner_id index="% index%" >

<input type="hidden" name="sku" value="%= mscsPage.HTMLEncode(sku) %"">
<input type="hidden" name="dept_id" value="%= Request("dept_id") %"">
<table border="0" cellspacing="2" cellpadding="2">
<tr>
<td>
<p><font size="6" face="arial"><b><%= mscsPage.HTMLEncode(name) %></b></font>

<% if not product_exists then %>
<p>The product you requested is currently not available.</p>
<% else %>
<form method=POST action="%= pageSURL("xt_orderform_edititem.asp")& &epartner_id=" &epartner_id index="% index%" >

<input type="hidden" name="sku" value="%= mscsPage.HTMLEncode(sku) %"">
<input type="hidden" name="dept_id" value="%= Request("dept_id") %"">
<table border="0" cellspacing="2" cellpadding="2">
<tr>
<td>
<p><font size="6" face="arial"><b><%= mscsPage.HTMLEncode(name) %></b></font>
<% if on_sale then %>
  <p><font size="4"><b>ON SALE! <%= MSCSDataFunctions.Money(sale_price) %></b></font></p>
  Regular Price: <%= MSCSDataFunctions.Money(list_price) %>
<% else %>
  <p><font size="4"><b><%= MSCSDataFunctions.Money(list_price) %></b></font></p>
<% end if %>

<p><%= mscsPage.HTMLEncode(description) %></p>

<p>&nbsp;</p>

<table border="0" width="100%"

  <tr>
    <td align="center" valign="top" width="100%"
      >&nbsp;
    </td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  </tr>
  <tr>
    <td align="center" valign="top" width="100%"
      >&nbsp;
    </td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  </tr>
  <tr>
    <td align="center" valign="top" width="100%"
      >&nbsp;
    </td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  </tr>
</table>

</form>
<% end if %>

<% REM get related products (if any): cmdTemp.CommandText = Replace("SELECT prod.sku, prod.name, deptprod.dept_id FROM fivelakes_promo_cross promo_cross, fivelakes_product prod, fivelakes_dept_prod deptprod WHERE promo_cross.sku = :1 and prod.sku = deptprod.sku and promo_cross.rel_sku = prod.sku", ":1", quoted_sku) Set rsRelated = Server.CreateObject("ADODB.Recordset") rsRelated.Open cmdTemp, , adOpenForwardOnly, adLockReadOnly
REM display up to 5 related products:
if Not rsRelated.EOF then
  <BR>
  <B>See Also</B>
  <% nRelated = 0
     set skuField = rsRelated("sku")
     set nameField = rsRelated("name")
     set dept_idField = rsRelated("dept_id")
     do while Not (rsRelated.EOF Or nRelated >= 5)
       <A HREF="<% = baseURL("product.asp") & mscsPage.URLShopperArgs("sku", skuField.value, "dept_id", dept_idField.value,"epartner_id=",epartner_id) %>"> <% = mscsPage.HTMLEncode(nameField.value) %> </A>
       nRelated = nRelated + 1
     rsRelated.MoveNext
   loop %>
<% end if %>

<%>

<--#! INCLUDE FILE="i_footer.asp" -->

</BODY>

</HTML>
product_edit.asp

```vbscript
<!--#INCLUDE FILE="include/Manager.asp" -->
<% REM   header: %>
<% cmdTemp.CommandText = Replace("SELECT * FROM fivelakes_product WHERE sku = ?", ", ", ", " & Replace(Request("sku"),"","" ) & """)
Set rsProduct = Server.CreateObject("ADODB.Recordset")
rsProduct.Open cmdTemp, , adOpenStatic, adLockReadOnly
pageTitle = "Edit Product' & rsProduct("name").value & "'
<% REM   body: %>
<% if Not rsProduct.EOF then %>
   <FORM METHOD="POST"
      ACTION="product_edit.asp">
      <INPUT TYPE="HIDDEN" NAME="Validate" VALUE="1">
      <TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0">
         <TR>
            <TD VALIGN="TOP">
               <TABLE CELLPADDING="5">
                  <TR>
                     <% REM  label: %>
                     <TH ALIGN="LEFT" VALIGN="TOP">
```
```
| Sku: | <input type="hidden" name="sku" value="\${mscsPage.HTMLEncode(rsProduct("sku"))}" />
|------|--------------------------------------------------
| Name: | <input type="text" name="name" value="\${mscsPage.HTMLEncode(rsProduct("name"))}" onChange="dirty = true" />
| Description: | <input type="text" name="description" value="\${mscsPage.HTMLEncode(rsProduct("description"))}" onChange="dirty = true" />
| List Price: | <input type="text" name="list_price" value="\${MSCSD ataFunctions.Money(rsProduct("list_price"))}" onChange="dirty = true" />
<table>
<thead>
<tr>
<th>Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image File</td>
<td><code>&lt;input type=&quot;text&quot; name=&quot;image_file&quot; value=&quot;&lt;%= mscsPage.HTMLEncode(Request(&quot;image_file&quot;)) %&gt;&quot; onchange=&quot;dirty = true&quot;/&gt;</code></td>
</tr>
<tr>
<td>Image Width</td>
<td><code>&lt;input type=&quot;text&quot; name=&quot;image_width&quot; value=&quot;&lt;%= mscsPage.HTMLEncode(rsProduct(&quot;image_width&quot;).value) %&gt;&quot; onchange=&quot;dirty = true&quot;/&gt;</code></td>
</tr>
<tr>
<td>Image Height</td>
<td><code>&lt;input type=&quot;text&quot; name=&quot;image_height&quot; value=&quot;&lt;%= mscsPage.HTMLEncode(rsProduct(&quot;image_height&quot;).value) %&gt;&quot; onchange=&quot;dirty = true&quot;/&gt;</code></td>
</tr>
<tr>
<td>Sale Price</td>
<td><code>&lt;%= mscsPage.HTMLEncode(rsProduct(&quot;sale_price&quot;).value) %&gt;</code></td>
</tr>
</tbody>
</table>
Appendix A: Lab and Practice Solution Files

<tr>
  <th align="LEFT" valign="TOP">
    Edition:
  </th>
  <td valign="TOP">
    <input type="text" size=32 name="edition" value="&lt;%= mscsPage.HTMLEncode(rsProduct("edition").value) %&gt;"
          onChange="dirty = true">
  </td>
</tr>
<tr>
  <th align="LEFT" valign="TOP">
    Author:
  </th>
  <td valign="TOP">
    <input type="text" size=32 name="author" value="&lt;%= mscsPage.HTMLEncode(rsProduct("author").value) %&gt;"
          onChange="dirty = true">
  </td>
</tr>
<tr>
  <th align="LEFT" valign="TOP">
    Category:
  </th>
  <td valign="TOP">
    <input type="text" size=32 name="category" value="&lt;%= mscsPage.HTMLEncode(rsProduct("category").value) %&gt;"
          onChange="dirty = true">
  </td>
</tr>

<td valign="TOP">
  <table border=0 cellpadding=0 cellspacing=0>
    <tr>
      <td valign="TOP">
        &lt;%= if Request("image_file").count &gt; 0 then %&gt;
          &lt;img src="/assets/product_images/&lt;%= mscsPage.HTMLEncode(Request("image_file")) %&gt;"&gt;
        &lt;/%= if %&gt;
    </td>
  </table>
</td>
Department: <BR>
<SELECT NAME="dept_id" SIZE="10" MULTIPLE>
<%}
<% cmdTemp.CommandText = Replace("SELECT dept_id FROM fivelakes_dept WHERE sku = ?", "," & Replace(rsProduct("sku").value,"",""))
Set rsPDept = Server.CreateObject("ADODB.Recordset")
rsPDept.Open cmdTemp, , adOpenStatic, adLockReadOnly
Do While Not rsPDept.EOF
selected = ""
if Not rsPDept.BOF then
%>
<BR>
<SELECT NAME="dept_id" SIZE="10" MULTIPLE>
<%}
<% cmdTemp.CommandText = Replace("SELECT dept_id FROM fivelakes_dept WHERE sku = ?", "," & Replace(rsProduct("sku").value,"",""))
Set rsPDept = Server.CreateObject("ADODB.Recordset")
rsPDept.Open cmdTemp, , adOpenStatic, adLockReadOnly
Do While Not rsPDept.EOF
selected = ""
if Not rsPDept.BOF then
%>
rsPDept.MoveNext
Do While Not rsPDept.EOF
if CInt(rsPDept("dept_id").value) = CInt(rsDept("dept_id").value)
then selected = " SELECTED"
rsPDept.MoveNext
Loop %>
NOTE: You must add the product to at least one department in order to make it visible for purchasing. **IMPORTANT** You must add Departments and select at least one department for each product to display them in the store.

PRODUCT not found.
<p>
<% end if %>
<br>

<% REM footer: %>
<!--#INCLUDE FILE="include/mgmt_footer.asp" -->
receipt.asp

<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<-- # INCLUDE FILE="i_shop.asp" -->
<-- # INCLUDE FILE="i_ePartner.asp" -->
<-- # INCLUDE FILE="i_util.asp" -->

<%
REM -- Create a storage object for the receipts
Set mscsReceiptStorage = UtilGetReceiptStorage()

REM -- retrieve receipt from storage using the shopper id and order id:
dim key(1), value(1)
key(0) = "order_id"
key(1) = "shopper_id"
value(0) = Request("order_id")
value(1) = mscsShopperID
on error resume next
set receipt = mscsReceiptStorage.LookupData(null, key, value)
on error goto 0
if IsEmpty(receipt) then
  items = null
  nitems = 0
else
  set items = receipt.items
  nitems = items.Count
end if
%

<HTML>
<HEAD>
  <TITLE><%= displayName %>: Receipt: <% = Request("order_id") %></TITLE>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>
<BODY
BGCOLOR="#FFFFFF"
TEXT="#000000"
LINK="#FF0000"
VLINK="#FF0000"
ALINK="#FF0000">
<!--#INCLUDE FILE="i_header.asp" -->
<% if nitems <> 0 then %>
<TABLE BORDER="0" CELLPADDING="2" CELLSPACING="1">
  <TR>
    </% if nitems <> 0 then %>
  </TABLE>
<!--#INCLUDE FILE="i_footer.asp" -->
</BODY>
<table>
<thead>
<tr>
<th>Label</th>
<th>Name</th>
</tr>
</thead>
</table>

**Receipt #** Request("order_id")<br><br>Ship To:<br><br>Bill To:<br><br>Date:
<table>
<thead>
<tr>
<th>Unit Price</th>
<th>Today’s Price</th>
<th>Qty</th>
<th>Extra Disc.</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

% REM row values: 
% for each row_item in items 

<table>
<thead>
<tr>
<th>TD WIDTH=&quot;75&quot; VALIGN=&quot;TOP&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%= mscsPage.HTMLEncode(row_item[_product_sku]) &gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%= mscsPage.HTMLEncode(row_item[_product_name]) &gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TD WIDTH=&quot;60&quot; VALIGN=&quot;TOP&quot; ALIGN=&quot;RIGHT&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%= MSCSDataFunctions.Money(row_item[_product_list_price]) &gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%= MSCSDataFunctions.Money(row_item[_iadjust_currentprice]) &gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TD WIDTH=&quot;30&quot; VALIGN=&quot;TOP&quot; ALIGN=&quot;CENTER&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%= row_item.quantity &gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%= MSCSDataFunctions.Money(row_item[_oadjust_discount]) &gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;%= MSCSDataFunctions.Money(row_item[_oadjust_adjustedprice]) &gt;</td>
</tr>
</tbody>
</table>

% next 
% REM divider: 

---
<TR>
<%>
ship_status = Request("ship_status")
if trim(ship_status) = "Yes" then
    shipping_message = "The order has been shipped"
else
    shipping_message = "The order has not yet been shipped"
end if
%</TR>

<TR BGCOLOR="#000000">
<TD>
<FONT COLOR="#FFFFFF">Shipping Status:</FONT>
</TD>
<TD>
<B><%=shipping_message%></B>
</TD>
</TR>

<TR>
<TH BGCOLOR="#000000" COLSPAN=8></TH>
</TR>

<TR>
<TH BGCOLOR="#000000" COLSPAN=8 HEIGHT=2></TH>
</TR>

<% REM show subtotal: %>
<TR>
<TD COLSPAN="5"></TD>
<TH BGCOLOR="#000000" VALIGN="TOP" ALIGN="RIGHT">
<FONT COLOR="#FFFFFF">Subtotal:</FONT>
</TH>
<TD VALIGN="TOP" ALIGN="RIGHT">
<B><% = MSCSDataFunctions.Money(receipt._oadjust_subtotal) %></B>
</TD>
</TR>

<% REM show shipping: %>
<TR>
<TD COLSPAN="5"></TD>
<TH BGCOLOR="#000000" VALIGN="TOP" ALIGN="RIGHT">
<FONT COLOR="#FFFFFF">Shipping:</FONT>
</TH>
<TD VALIGN="TOP" ALIGN="RIGHT">
<B><% = MSCSDataFunctions.Money(receipt._shipping_total) %></B>
</TD>
</TR>

<% REM show handling: %>
<TR>
<TD COLSPAN="5"></TD>
<TH BGCOLOR="#000000" VALIGN="TOP" ALIGN="RIGHT">
<FONT COLOR="#FFFFFF">Handling:</FONT>
</TH>
<TD VALIGN="TOP" ALIGN="RIGHT">
<% = MSCSDataFunctions.Money(receipt._handling_total) %>
</TD>
</TR>
Appendix A: Lab and Practice Solution Files

```html
<br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>}
receipts.asp

```html
<%@ LANGUAGE=vbscript  enable session state=false LCID=1033 %>  

<% Response.ExpiresAbsolute=DateAdd("yyyy", -10, Date) %>

<-- # INCLUDE FILE="i_shop.asp" -->
<-- # INCLUDE FILE="i_ePartner.asp" -->
<-- # INCLUDE FILE="i_util.asp" -->

<HTML>
<HEAD>
<TITLE><%= displayName %>: Order History</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#FF0000" VLINK="#FF0000" ALINK="#FF0000">

<!--# INCLUDE FILE="i_header.asp" -->

<%
sqlText = Replace(MSCSQueryMap.receipts_for_shopper.SQLCommand,":1","'
& mscsShopperID & '")
Set rsReceipts = MSCS.Execute (sqlText, nReceipts, adCmdText)
%>

<TABLE WIDTH="400" BORDER="0" CELLPADDING="2" CELLSPACING="2">
  <TR>
    <TD COLSPAN="3" ALIGN="CENTER" BGCOLOR="#000000">
      <FONT COLOR="#FFFFFF" SIZE="5">
        Order History
      </FONT>
    </TD>
  </TR>
  <TR>
    <TH BGCOLOR="#000000" ALIGN="LEFT">Order #</TH>
    <TH BGCOLOR="#000000" ALIGN="LEFT">Date</TH>
    <TH BGCOLOR="#000000" ALIGN="RIGHT">Amount</TH>
  </TR>
```
<table>
<thead>
<tr>
<th>Order ID</th>
<th>Date Entered</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No receipts found.
shipping.asp

<<replace this page with new code after Julie updates the project.>>
xt_orderform_additem.asp

```vbscript
<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<-- # INCLUDE FILE="i_shop.asp" -->
<-- # INCLUDE FILE="i_ePartner.asp" -->
<-- # INCLUDE FILE="i_util.asp" -->

<% function OrderFormAddItem(byVal orderFormStorage, byVal shopperID)
    Set mscsOrderForm = UtilGetOrderForm(mscsOrderFormStorage, created)
    REM -- retrieve quantity:
    product_qty = mscsPage.RequestNumber("qty", "1", 1, 999)
    if IsNull(product_qty) then
        product_qty = 1
    end if

    REM -- retrieve sku and dept_id:
    sku = mscsPage.RequestString("sku")
    quoted_sku = "'" & Replace(sku,"'","''") & "'"
    dept_id = mscsPage.RequestNumber("dept_id")

    REM -- retrieve product:
    sqlText = MSCSQueryMap.product_info.SQLCommand
    sqlText = Replace(sqlText, ",", quoted_sku, 1, 1)
    sqlText = Replace(sqlText, ",", dept_id, 1, 1)
    cmdTemp.CommandText = sqlText
    Set rsProduct = Server.CreateObject("ADODB.Recordset")
    rsProduct.Open cmdTemp, , adOpenStatic, adLockReadOnly
    list_price = rsProduct("list_price").value
    name = rsProduct("name").value
    rsProduct.Close

    REM -- add item to order form
    Set item = mscsOrderForm.AddItem(sku, product_qty, list_price)
    item.name = name
    item.list_price = list_price
    item.dept_id = dept_id

    MSCS.Close

    REM -- commit order form back to storage:
    Call UtilPutOrderForm(orderFormStorage, mscsOrderForm, created)

    OrderFormAddItem = true
end function
```
Set mcsOrderFormStorage = UtilGetOrderFormStorage()

success = OrderFormAddItem(mcsOrderFormStorage, mcsShopperID)
call Response.Redirect("basket.asp?" & mcsPage.URLShopperArgs() & "&epartner_i d=" & epartner_id)

%>
xt_orderform_clearitems.asp

```vbscript
<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<-- # INCLUDE FILE="i_shop.asp" -->
<-- # INCLUDE FILE="i_ePartner.asp" -->
<-- # INCLUDE FILE="i_util.asp" -->

<%
REM -- Create the order form storage
set mscsOrderFormStorage = UtilGetOrderFormStorage()

REM Get the order form
set mscsOrderForm = UtilGetOrderForm(mscsOrderFormStorage, created)

if mscsOrderForm.Items.Count > 0 then
    call mscsOrderForm.ClearItems()

    call mscsOrderFormStorage.CommitData(NULL, mscsOrderForm)
end if

call Response.Redirect("basket.asp?" & mscsPage.URLShopperArgs() & "&partner_id=" & epartner_id) %>
```
xt_orderform_delitem.asp

<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<!--# INCLUDE FILE="i_shop.asp" -->
<!--# INCLUDE FILE="i_ePartner.asp" -->
<!--# INCLUDE FILE="i_util.asp" -->

<%
REM Create a storage object for the order forms
Set mscsOrderFormStorage = UtilGetOrderFormStorage()

REM Get the order form
set mscsOrderForm = UtilGetOrderForm(mscsOrderFormStorage, created)

REM Get the index of the item to delete
if mscsOrderForm.Items.Count > 0 then
    index = mscsPage.RequestNumber("index", NULL, 0, mscsOrderForm.Items.Count - 1)
    if Not IsNull(index) then
        call mscsOrderForm.Items.Delete(index)
        call mscsOrderFormStorage.CommitData(NULL, mscsOrderForm)
    end if
end if

%>
xt_orderform_edititem.asp

```vbscript
<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>
<% Response.Buffer=true %>

<!--#INCLUDE FILE="i_shop.asp" -->
<!--#INCLUDE FILE="i_ePartner.asp" -->
<!--#INCLUDE FILE="i_util.asp" -->

<%
    function OrderFormAddItem(byVal orderFormStorage, byVal shopperID)
        Set mscsOrderForm = UtilGetOrderForm(mscsOrderFormStorage, created)
        if mscsOrderForm.Items.Count > 0 then
            index=Request.QueryString("index")
            call mscsOrderForm.Items.Delete(index)
            call mscsOrderFormStorage.CommitData(NULL, mscsOrderForm)
        end if
        REM -- retrieve quantity:
        product_qty = mscsPage.RequestNumber("qty", "1", 1, 999)
        if IsNull(product_qty) then
            product_qty = 1
        end if
        REM -- retrieve sku and dept_id:
        sku = mscsPage.RequestString("sku")
        quoted_sku = "'" & Replace(sku,"'","''") & "'
        dept_id = mscsPage.RequestNumber("dept_id")

        REM -- retrieve product:
        sqlText = MSCSQueryMap.product_info.SQLCommand
        sqlText = Replace(sqlText, "?", quoted_sku, 1, 1)
        sqlText = Replace(sqlText, "?", dept_id, 1, 1)
        cmdTemp.CommandText = sqlText
        Set rsProduct = Server.CreateObject("ADODB.Recordset")
        rsProduct.Open cmdTemp, , adOpenStatic, adLockReadOnly
        list_price = rsProduct("list_price").value
        name = rsProduct("name").value
        rsProduct.Close

        REM -- add item to order form
        set item = mscsOrderForm.AddItem(sku, product_qty, list_price)
        itemName = name
        itemlist_price = list_price
        item dept_id = dept_id

        MS cs.Close
```
REM -- commit order form back to storage:
  Call UtilPutOrderForm(orderFormStorage, mscsOrderForm, created)

  OrderFormAddItem = true
end function

Set mscsOrderFormStorage = UtilGetOrderFormStorage()

success = OrderFormAddItem(mscsOrderFormStorage, mscsShopperID)
Appendix A: Lab and Practice Solution Files

xt_orderform_editquantities.asp

<%@ LANGUAGE=vbscript enable sessi onstate=false LCID=1033 %>

<--# NCLUDE FILE="i_shop.asp" -->
<--# NCLUDE FILE="i_ePartner.asp" -->
<--# NCLUDE FILE="i_util.asp" -->

<%
REM Create a storage object for the order forms
Set mcsOrderFormStorage = UtilGetOrderFormStorage()
REM Get the order form
set mcsOrderForm = UtilGetOrderForm(mcsOrderFormStorage, created)

strError = ""
if Not IsEmpty(mcsOrderForm) then
    set items = mcsOrderForm.Items
    for index = mcsOrderForm.Items.Count - 1 to 0 step -1
        set item = items(index)
        quantity = mcsPage.RequestNumber("qty_" &
            CStr(index), item quantity, 0, 999)
        if IsNull(quantity) then
            strError = "nonnumber"
        else
            if quantity = 0 then
                call mcsOrderForm.Items.Delete(index)
            else
                item quantity = quantity
                call MSCSOrderFormStorage.CommitData(null,
                    mcsOrderForm)
            end if
        end if
    next
end if

if strError = "" then
    pageRedirect = "basket.asp" & mcsPage.URLShopperArgs()
else
    pageRedirect = "basket.asp" & mcsPage.URLShopperArgs("error", strError,"epartner_id="
        epartner_id)
end if

Response.Redirect(pageRedirect)
%>
xt_orderform_prepare.asp

<%@ LANGUAGE=vbscript enablesessionstate=false LCID=1033 %>

<-- # NCLUDE FILE="i_shop.asp" -->
<-- # NCLUDE FILE="i_ePartner.asp" -->
<-- # NCLUDE FILE="i_util.asp" -->

<% function OrderFormPrepareArgs(byRef orderForm, byRef errorList)

orderForm.epartner_ID=request.QueryString("epartner_id")

REM -- shipping method:
orderForm.shipping_method = mscsPage.RequestString("shipping_method")

REM -- ship to:
ship_to_name = mscsPage.RequestString("ship_to_name", null, 1, 100)
if IsNull(ship_to_name) then
    errorList.Add("Ship to name must be a string between 1 and 100 characters")
else
    orderForm.ship_to_name = ship_to_name
    orderForm.bill_to_name = orderForm.ship_to_name
end if

ship_to_street = mscsPage.RequestString("ship_to_street", null, 1, 100)
if IsNull(ship_to_street) then
    errorList.Add("Ship to street must be a string between 1 and 100 characters")
else
    orderForm.ship_to_street = ship_to_street
    orderForm.bill_to_street = orderForm.ship_to_street
end if

ship_to_city = mscsPage.RequestString("ship_to_city", null, 1, 100)
if IsNull(ship_to_city) then
    errorList.Add("Ship to city must be a string between 1 and 100 characters")
else
    orderForm.ship_to_city = ship_to_city
    orderForm.bill_to_city = orderForm.ship_to_city
end if

ship_to_state = mscsPage.RequestString("ship_to_state", null, 1, 100)
if IsNull(ship_to_state) then
    errorList.Add("Ship to state must be a string between 1 and 100 characters")
else
    orderForm.ship_to_state = ship_to_state
    orderForm.bill_to_state = orderForm.ship_to_state
end if

else

orderForm.ship_to_state = ship_to_state
orderForm.bill_to_state = orderForm.ship_to_state
end if

ship_to_zip = mscsPage.RequestString("ship_to_zip", null, 1, 100)
if IsNull(ship_to_zip) then
    errorList.Add("Ship to zip must be a string between 1 and 100 characters")
else
    orderForm.ship_to_zip = ship_to_zip
    orderForm.bill_to_zip = orderForm.ship_to_zip
end if

ship_to_country = mscsPage.RequestString("ship_to_country", null, 1, 100)
if IsNull(ship_to_country) then
    errorList.Add("Ship to country must be a string between 1 and 100 characters")
else
    orderForm.ship_to_country = ship_to_country
    orderForm.bill_to_country = orderForm.ship_to_country
end if

ship_to_phone = mscsPage.RequestString("ship_to_phone", null, 1, 100)
if IsNull(ship_to_phone) then
    errorList.Add("Ship to phone must be a string between 1 and 100 characters")
else
    orderForm.ship_to_phone = ship_to_phone
    orderForm.bill_to_phone = orderForm.ship_to_phone
end if

ship_to_email = mscsPage.RequestString("ship_to_email", null, 1, 100)
if IsNull(ship_to_email) then
    errorList.Add("Ship to email must be a string between 1 and 100 characters")
else
    orderForm.ship_to_email = ship_to_email
    orderForm.bill_to_email = orderForm.ship_to_email
end if

REM Retrieve tax_rate and code from FiveLakes_country table
quoted_country = "'" & Replace(ship_to_country,"'","''") & "'
REM - add quotes
sqlText = MSCSQueryMap.country_tax.SQLCommand
sqlText = Replace(sqlText,":1",quoted_country)
sqlText = Replace(sqlText,":2",quoted_country)
cmdTemp.CommandText = sqlText
set rsccountry_tax = Server.CreateObject("ADODB.Recordset")
rsccountry_tax.Open cmdTemp, , adOpenForwardOnly, adLockReadonly
if not rsccountry_tax.EOF then
    orderForm.ship_to_country = rsccountry_tax("country")
else
    orderForm.ship_to_country = ""
orderform.tax_rate = rsfunction_tax("tax_rate")
end if
end function

REM Create a dictionary to store errors
Set errorList = Server.CreateObject("Commerce.SimpleList")

REM Get the order form storage
Set mscsOrderFormStorage = UtilGetOrderFormStorage()

REM Get the order form
Set mscsOrderForm = UtilGetOrderForm(mscsOrderFormStorage, created)

REM Retrieve the args from the form
Call OrderFormPrepareArgs(mscsOrderForm, errorList)

if errorList.Count > 0 then
  <!--#INCLUDE FILE="i_error.asp" -->
  <%
else
  call UtilPutOrderForm(mscsOrderFormStorage, mscsOrderForm, created)

  call Response.Redirect("payment.asp? epartner_id=" &
  epartner_id & "; use_form", mscsPage.RequestString("use_form", 0))
end if
%>
function OrderFormPurchaseArgs(byRef orderForm, byRef errorList)
REM -- cc info:
cc_name = mscsPage_RequestString("cc_name", null, 1, 100)
if IsNull(cc_name) then
    errorList.Add("Credit card name must be a string between 1 and 100 characters")
else
    orderForm.cc_name = cc_name
end if

cc_type = mscsPage_RequestString("cc_type", null, 1, 100)
orderForm.cc_type = cc_type
if IsNull(cc_type) then
    errorList.Add("Credit card type must be a string between 1 and 100 characters")
else
    orderForm.cc_type = cc_type
end if

cc_number = mscsPage_RequestString("_cc_number", null, 13, 19)
if IsNull(cc_number) then
    errorList.Add("Credit card number must be a string between 13 and 19 characters")
else
    orderForm._cc_number = cc_number
end if

cc_expmonth = mscsPage_RequestNumber("_cc_expmonth", null, 1, 12)
if IsNull(cc_expmonth) then
    errorList.Add("Expiration month must be a number between 1 and 12")
else
    orderForm._cc_expmonth = cc_expmonth
end if

if IsNull(cc_expyear) then
    errorList.Add("Expiration year must be a number between 1997 and 2003")
else
    orderForm._cc_expyear = cc_expyear
end if
REM -- bill to:
bill_to_name = mscsPage.RequestString("bill_to_name", null, 1, 100)
if IsNull(bill_to_name) then
    errorList.Add("Ship to name must be a string between 1 and 100 characters")
else
    orderForm.bill_to_name = bill_to_name
end if

bill_to_street = mscsPage.RequestString("bill_to_street", null, 1, 100)
if IsNull(bill_to_street) then
    errorList.Add("Ship to street must be a string between 1 and 100 characters")
else
    orderForm.bill_to_street = bill_to_street
end if

bill_to_city = mscsPage.RequestString("bill_to_city", null, 1, 100)
if IsNull(bill_to_city) then
    errorList.Add("Ship to city must be a string between 1 and 100 characters")
else
    orderForm.bill_to_city = bill_to_city
end if

bill_to_state = mscsPage.RequestString("bill_to_state", null, 1, 100)
if IsNull(bill_to_state) then
    errorList.Add("Ship to zip must be a string between 1 and 100 characters")
else
    orderForm.bill_to_state = bill_to_state
end if

bill_to_zip = mscsPage.RequestString("bill_to_zip", null, 1, 100)
if IsNull(bill_to_zip) then
    errorList.Add("Ship to zip must be a string between 1 and 100 characters")
else
    orderForm.bill_to_zip = bill_to_zip
end if

bill_to_country = mscsPage.RequestString("bill_to_country", null, 1, 100)
if IsNull(bill_to_country) then
    errorList.Add("Ship to country must be a string between 1 and 100 characters")
else
    orderForm.bill_to_country = bill_to_country
end if

bill_to_phone = mscsPage.RequestString("bill_to_phone", null, 1, 100)
if IsNull(bill_to_phone) then
    errorList.Add("Ship to phone must be a string between 1 and 100 characters")
else
    orderForm.bill_to_phone = bill_to_phone
end if
else
  orderForm.bill_to_phone = bill_to_phone
end if

OrderFormPurchaseArgs = true
end function

function OrderFormPurchase(byRef errorList)
  OrderFormPurchase = null

  Set rsOrderID = Server.CreateObject("ADODB.Recordset")
  REM Extract Order ID value from the table
  cmdTemp.CommandText = "select NewOrderID from FiveLakes_orderid"
  set rsOrderID = cmdTemp.Execute
  New_oid = rsOrderID("NewOrderID")

  temp_val = Int(Mid(New_oid,7)) + Int("1")

  if len(month(now())) = 2 then
    next_order_id = month(now()) & year(now()) & CStr(temp_val)
  else
    next_order_id = "0" & month(now()) & year(now()) & CStr(temp_val)
  end if

  cmdTemp.CommandText = "update FiveLakes_orderid set NewOrderID = :1"
  next_order_id = "'" & next_order_id & "'
  cmdTemp.CommandText = Replace(cmdTemp.CommandText, ":1", next_order_id)
  set rsOrderID = cmdTemp.Execute

  REM Create a storage object for the order form
  Set mscsOrderFormStorage = UtilGetOrderFormStorage()

  REM Retrieve order from the storage
  Set mscsOrderForm = UtilGetOrderForm(mscsOrderFormStorage, created)

  REM Storing the newly created ID
  mscsOrderForm.order_id = New_oid

  REM Retrieve args from form
  Call OrderFormPurchaseArgs(mscsOrderForm, errorList)

  REM If the order form has no items, add an error
  if mscsOrderForm.Items.Count = 0 then
    errorList.Add("No items to order.")
  end if

  if errorList.Count > 0 then
    REM Save changes to the order form so far

call UtilPutOrderForm(mscsOrderFormStorage, mscsOrderForm, created)

exit function
end if

REM Set the verify with flags onto the orderform
call mscsPage.ProcessVerifyWith(mscsOrderForm)

REM Create the basic pipe context
set mscsPipeContext = UtilGetPipeContext()

REM Run the plan
errorLevel = UtilRunPipe("plan.pcf", mscsOrderForm, mscsPipeContext)

REM -- Finally if no errors, run the actual purchase
REM -- Create a transacted pipeline for this execution
if mscsOrderForm[\_Basket_Errors].Count = 0 and mscsOrderForm[\_Purchase_Errors].Count = 0 and errorLevel = 1 then

REM Create the receipt storage
Set mscsReceiptStorage = UtilGetReceiptStorage()

REM Add the receipt storage into the pipe context...the Save Receipt component uses it
Set mscsPipeContext.ReceiptStorage = MSCSSeceiptStorage

REM Run the transacted pipe
cmdTemp.CommandText = Replace(MSCSQueryMap.email_Search.SQLCommand, ":1", Replace(mscsOrderForm.ship_to_email, ":", ","))
Set rsEmail = Server.CreateObject("ADODB.Recordset")
rsEmail.Open cmdTemp, , adOpenKeyset, adLockReadOnly
if not rsEmail.EOF then
    errorLevel = UtilRunTxPipe("purchaseupdate.pcf", mscsOrderForm, mscsPipeContext)
else
    errorLevel = UtilRunTxPipe("purchaseinsert.pcf", mscsOrderForm, mscsPipeContext)
end if
end if

if mscsOrderForm[\_Basket_Errors].Count > 0 then
    REM -- goto basket to show errors
    Response.redirect "basket.asp?" & mscsPage.URLShopperArgs()
    Response.End
end if

if mscsOrderForm[\_Purchase_Errors].Count > 0 or errorLevel > 1 then
    if mscsOrderForm[\_Purchase_Errors].Count > 0 then
for each errorStr in mscsOrderForm._Purchase_Errors
    errorList.Add(errorStr)
next
else
    errorList.Add("Unable to complete purchase at this time")
end if
OrderFormPurchase = null
exit function
end if

REM Save the order id before we delete it
order_id = mscsOrderForm.order_id

REM Purchase was successful....delete the order form from the storage
call MSCSOrderFormStorage.DeleteData(null, mscsOrderForm)

REM Return the order id
OrderFormPurchase = order_id
end function

Set errorList = Server.CreateObject("Commerce.SimpleList")

order_id = OrderFormPurchase(errorList)
if errorList.Count > 0 then

    <!--#INCLUDE FILE="i_error.asp" -->
<%>
else
    call Response.Redirect("confirmed.asp?" & mscsPage.URLShopperArgs("order_id", order_id, "epartner_id=", epartner_id))
end if
<%>