Ingres® 2006 Release 2

Web Deployment Option User Guide

INGRES®

February 2007

Contents

Chapter 1: Introduction

What You Need to Know 1-1
Where to Go from Here 1-1

Chapter 2: Getting Started

Installing the HTTP Server 2-1
Configuring the HTTP Server 2-1
    What the Web Server Needs to Know 2-2
    Adding Virtual Directories 2-2
    Enabling the Native HTTP Server Extensions 2-3
    Rebooting Windows 2-3
Microsoft Internet Information Server (IIS) 2-3
    Environment (IIS) 2-4
    Virtual Directories (IIS) 2-8
    ICE File Type (IIS) 2-11
    Using Your Web Server as a Windows Service 2-11
Apache Web Server 2-11
    Environment (Apache) 2-12
    Virtual Directories (Apache) 2-12
    ICE File Type (Apache) 2-12
    Using Your Web Server as a Windows Service 2-14
Setting Up Your ICE Server 2-14
    Web Server Document Directory 2-15

Chapter 3: Understanding the Web Deployment Option

Overview 3-1
Users 3-2
Architecture 3-3
    Web Site Components 3-4
    Web Browser 3-5
    Web Deployment Option Client 3-5
    ICE Server 3-6
    Information Systems 3-6

Chapter 4: Managing the Web Deployment Option

Accessing Web Deployment Option Information 4-1
Managing Security 4-3
- Web Users 4-3
- Database Users 4-5
- Database Connections 4-6
- Roles 4-7
- Profiles 4-8

Managing Server Information 4-9
- Session Groups 4-9
- Locations 4-10
- ICE Server Variables 4-11

Managing Business Units 4-13
- Business Units 4-13
- Documents, Pages, and Facets 4-15
- Role Access Definitions 4-16
- Web User Access Definitions 4-17
- Associating a Location with a Business Unit 4-18

Monitoring Web Deployment Option Information 4-19
Shutting Down 4-19

Chapter 5: Using the Macro Language

Web Deployment Option XML Tag Set 5-1
- Web Deployment Option XML Macro Tag Format 5-2
- Web Deployment Option Macro Tags 5-2
- Tag Hierarchy 5-3

Macro Tags 5-4
- <i3ce_commit> Tag 5-4
- <i3ce_declare> Tag 5-4
- <i3ce_extend> Tag 5-6
- <i3ce_function> Tag 5-8
- <i3ce_if> Tag 5-9
- <i3ce_include> Tag 5-11
- <i3ce_query> Tag 5-13
- <i3ce_rollback> Tag 5-19
- <i3ce_switch> Tag 5-19
- <i3ce_var> Tag 5-20

Macro Statements 5-21
- Macro Statement Format 5-21
- Macro Keywords 5-22

Macro Keywords 5-22
COMMIT Keyword 5-22
DECLARE Keyword 5-23
FUNCTION Keyword 5-25
IF Keyword 5-27
INCLUDE Keyword 5-28
ROLLBACK Keyword 5-30
SQL Keyword 5-31
SWITCH Keyword 5-39
VAR Keyword 5-41

Chapter 6: Creating Web Applications: An Example

Before You Begin 6-2
A Tour of the Plays Application 6-2
Plays Welcome Page 6-3
Plays Login Page 6-4
Automatic Declaration Page 6-5
Plays Home Page 6-6
Plays View Options 6-7
Globe Boutique 6-10
Creating Application Directories 6-12
Creating Directories for Non-Web Deployment Option Registered Files 6-12
Creating Directories for Web Deployment Option-Registered Files 6-13
Creating Application Files 6-13
Creating the Starting Application Page 6-14
Creating the Welcome Page and Facets 6-15
Creating the Remaining Pages and Facets 6-16
Using Style Sheets 6-17
Gaining Access to Web Deployment Option Information 6-18
Registering Your Files and Location 6-18
Creating a Session Group 6-18
Setting Up Public Files 6-19
Creating a Server Location for Secured Pages 6-19
Creating a Business Unit 6-20
Associating the Server Location with the Business Unit 6-21
Associating Pages with the Business Unit 6-21
Associating Facets with the Business Unit 6-22
Creating a Database Connection 6-24
Creating a Profile 6-24
Creating a Role 6-25
Associating a Role with a Profile 6-26

Associating a Database Connection with a Profile 6-27
Associating a Role with a Business Unit 6-27
Designing a Data Browsing Application 6-28
Creating a Welcome Page 6-28
Creating a Login Page 6-29
Creating a Home Page 6-31
Creating a User Account Automatically 6-32
Displaying All Table Rows 6-33
Displaying All Table Rows with Wrapping 6-40
Creating an Automatically-Generated Selector Control 6-43
Displaying a Subset of Table Rows by Selector 6-45
Creating Automatically-Generated Hyperlinks 6-47
Displaying a Subset of Table Rows by Hyperlink 6-48
Creating Graphical Hyperlinks 6-50
Creating Switch Image Links 6-52
Designing an Internet Shopping Application 6-54
The Globe Boutique Home Page 6-54
Creating the Tables for the Globe Boutique Application 6-54
Creating the New Order Procedure 6-55
Creating the New Order Extension Header File 6-55
Creating the New Order Extension 6-56
Building the New Order Extension 6-58
Displaying an Item Description 6-62
Adding an Item to the Shopping Bag 6-64
Displaying Shopping Bag Contents 6-67
Confirming an Order 6-70
Rolling Back a Transaction 6-72
Plays Tutorial Application Data 6-73

Chapter 7: Using the C API

Web Deployment Option C API Reference 7-1
ICE_C_Close() Function 7-1
ICE_C_Connect() Function 7-2
ICE_C_Disconnect() Function 7-3
ICE_C_Execute() Function 7-3
ICE_C_Fetch() Function 7-5
ICE_C_GetAttribute() Function 7-6
ICE_C_Initialize() Function 7-7
ICE_C_LastError() Function 7-8
ICE_STATUS Data Type 7-8
ICE_C_CLIENT Structure 7-9

ICE_C_PARAMS Structure 7-9
Sample C API for Web Deployment Option 7-11

Chapter 8: Writing ICE Server Extension Functions

Defining an Initialization Function 8-1
Providing a Function Description 8-2
Defining Your Extension Function 8-3
Calling an Extension Function from a Web Page 8-4
Sample Extension Library 8-5
Plays Example 8-5

Appendix A: XML Primer

XML Overview A-1
Extensible A-1
Complementary with HTML A-2
XML Syntax A-2
All Elements Have A Closing Tag A-2
XML Tags Are Case Sensitive A-2
XML Elements Must Be Properly Nested A-2
XML Documents Must Have a Root Tag A-3
Attribute Values Must Always Be In Quotation Marks A-3
XML Example A-3
XML and Web Deployment Option Queries A-4

Appendix B: HTML Primer

The Development of HTML B-1
Anatomy of an HTML Document B-1
Elements Used by Web Deployment Option B-5
Elements Generated by Web Deployment Option B-7
Accessing Web Deployment Option Pages B-7

Appendix C: Reserved Words

Reserved Words C-1

Appendix D: ICE Server Functions

Security Functions D-1
DBUser() Function D-1
Database() Function D-2
Role() Function D-2
User() Function D-2
User_Role() Function D-3
User_Database() Function D-4
Profile() Function D-4
Profile_Role() Function D-5
Profile_Database() Function D-5
Business Unit Functions D-6
Unit() Function D-6
Unit_Role() Function D-6
Unit_User() Function D-7
Unit_Location() Function D-7
Unit_Copy() Function D-8
Document() Function D-8
Document_Role() Function D-9
Document_User() Function D-10
Session_Grp() Function D-11
Server Function D-11
ICE_Locations() Function D-11
Monitoring Functions D-12
Active_Users() Function D-12
ICE_Users() Function D-13
ICE_User_Transactions() Function D-13
ICE_User_Cursors() Function D-14
ICE_Cache() Function D-14
ICE_Connect_Info() Function D-15
Additional Functions D-16
TagToString() Function D-16
Dir() Function D-16
GetVariables() Function D-16

Appendix E: Using an XML Authoring Tool

Starting XMetaL E-1
Creating a New Document E-2
Building Macro Elements E-4
Using the XMetaL Environment E-4

Index

Chapter 1: Introduction

Ingres® Web Deployment Option provides the foundation for Internet-based electronic commerce. It allows developers to build World Wide Web (Web) applications that can access enterprise-wide corporate data.

This guide provides information on how to configure and manage your Web Deployment Option environment. It also walks you through the development of a sample application, from start to finish.

What You Need to Know

If you are a Web Deployment Option application developer, this guide assumes that you are familiar with:

- Basic HTML/XML form development concepts
- Embedded SQL for setting up queries on a Web page
- SQL for executing SQL statements from a Web page
- C/C++ programming language

If you are a Web Deployment Option administrator, you should be familiar with the following:

- Ingres network administration
- Ingres system administration
- Web server administration
- Database administration

In addition, you should be familiar with Windows, including terminology, navigational techniques, and working with standard items, such as menus and dialogs. Some knowledge of networking concepts and the Web is also assumed.

Where to Go from Here

To gain a better understanding of the Web Deployment Option, all users—both novice and advanced—should read the remainder of this guide, including "Chapter 6: Creating Web Applications: An Example," which shows you how to design Web applications that interact with Ingres databases.

Chapter 2: Getting Started

This chapter describes post-installation steps for Web Deployment Option. It shows you how to set up your Web (HTTP) server and your Web Deployment Option application environment.

Note: You should already have installed Web Deployment Option as part of your Ingres installation, and started the ICE Server. For more information about starting servers, see the chapter "Managing Your System and Monitoring Performance" in the *System Administrator Guide*.

HTTP Server

Before you can run a Web Deployment Option application, you must have one of the following Web (HTTP) servers installed:

- Microsoft Internet Information Server
- Apache Web Server

For installation instructions, see the documentation for your particular server.

After testing to see that the web server is working properly, you must configure the web server to work with Web Deployment Option.

Configuring the HTTP Server

You must configure your web server so that it will communicate with the Web Deployment Option client. Many of the Web Deployment Option client files—including the binary file and images files used in your applications—are located under the II_SYSTEM directory structure. To enable the referencing of these files, you must include additional virtual directories in the web server configuration.

What the Web Server Needs to Know

Your web server must know how to locate and invoke the Web Deployment Option custom extension appropriate for it. You must provide the following two pieces of information to the server so that it can invoke the Web Deployment Option system:

- Location of the extension file (Dynamic Link Library or Shared Object)
- Location of the public files (Web Deployment Option picture directory)

In addition, it may be necessary on some systems to ensure that the environment is correctly set and passed on. You should ensure that the web server process can access the environment variable II_SYSTEM and that the appropriate additions have been made to your operating system search paths for programs and shared or dynamic libraries.

Adding Virtual Directories

To let the web server know the location of the above files, the following virtual directory aliases must be added to your web server configuration:

- /ice-bin
- /iceimages

The virtual directories are mapped to the actual directories where the files reside.

Directory /ice-bin

The Web Deployment Option client binaries (extension files) are located in the following directory:

Windows

%II_SYSTEM%\ingres\ice\bin\<*web server directory*>

UNIX

$II_SYSTEM/ingres/ice/bin/<*web server directory*>

where <*web server directory*> is the name of the subdirectory where the server extension is located. For the supported server extensions, the possible subdirectories are microsoft and apache.

The HTTP server invokes the requested binary when it is specified in the URL or from a Web page. For more information on the Web Deployment Option client, see "Chapter 3: Understanding the Web Deployment Option."

Directory /iceimages

The public image files are located in the following directory:

Windows

%II_SYSTEM%\ingres\ice\images

UNIX

$II_SYSTEM/ingres/ice/images

This directory is used for the non-secured, initial images presented in the sample Plays tutorial application, in addition to the My_Plays application that you will create.

Enabling the Native HTTP Server Extensions

In addition to the virtual directories, you must configure your web server to enable use of the Web Deployment Option native HTTP server extensions. Instructions are in the following sections.

Rebooting Windows

Important! After you have installed and configured the web server, you must reboot Windows systems. This ensures that Ingres is included in the path for the Windows Service Control Manager.

Value of II_System

This chapter uses the notation *<II_SYSTEM>*, which refers to the value of the II_SYSTEM environment variable for your Ingres installation. You must know this value to complete certain configuration tasks.

To learn this value, enter the following at a command prompt:

```
ingprenv II_SYSTEM
```

Microsoft Internet Information Server (IIS)

To configure IIS to work with Web Deployment Option, complete the following tasks:

1. Create a user for Web Deployment Option
2. Configure IIS
3. Reboot

Environment (IIS)

Windows systems allow a global environment to be set for all processes on the machine.

The Web Deployment Option installation should have set the following environment variables, which allow IIS to connect to the ICE Server:

- II_SYSTEM
- PATH

The PATH variable should include entries for Ingres binary and dynamic link library directories.

Create a User for Web Deployment Option

A user must be created that has both anonymous Web site and anonymous application access.

To create a local operating system user for Web Deployment Option

1. Start the Computer Management Console (CMC) by selecting Start, Run. Type the command:

   ```
   mmc C:\WINDOWS\system32\compmgmt.msc
   ```

   a. Create a new user. IIS Web Deployment Option extensions and application pool will be identified by this user.

   b. Give this user a password that cannot be changed and never expires.

   c. Make this user a member of the Guests and IIS_WPG groups.

2. Start the Local Security Console (LSC) by selecting Start, Run. Type the command:

   ```
   mmc C:\WINDOWS\system32\secpol.msc
   ```

   a. Under Local Policies, User Rights Assignments, add the previously created user to the rights "Replace process-level token" and "Adjust memory quotas for a process."

How You Configure IIS for Web Deployment Option

Windows

You configure the Microsoft Internet Information Server from the Internet Service Manager.

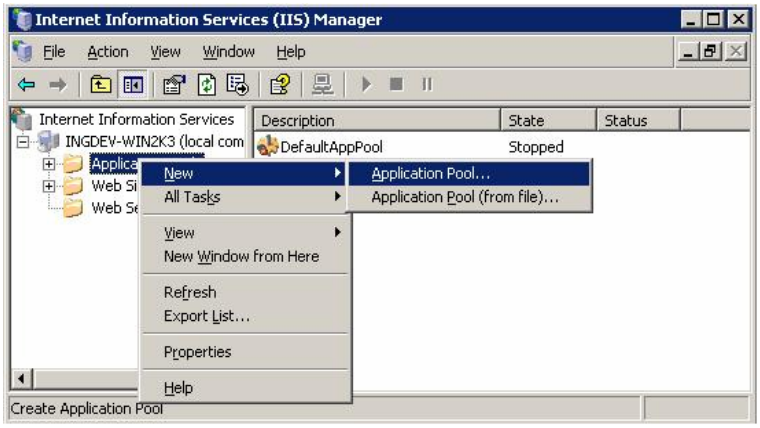
Both the CGI and ISAPI standards are supported for IIS.

The overall procedure is as follows:

1. Create a named application pool and assign WDO user to identity.
2. Create a separate web site.
3. Add ice_index.html to Documents property for the web site.
4. Add virtual directories for ice-bin and iceimages. Set application settings for the ice-bin directory.
5. Create new Web services extensions oiice.exe and oiice.dll.

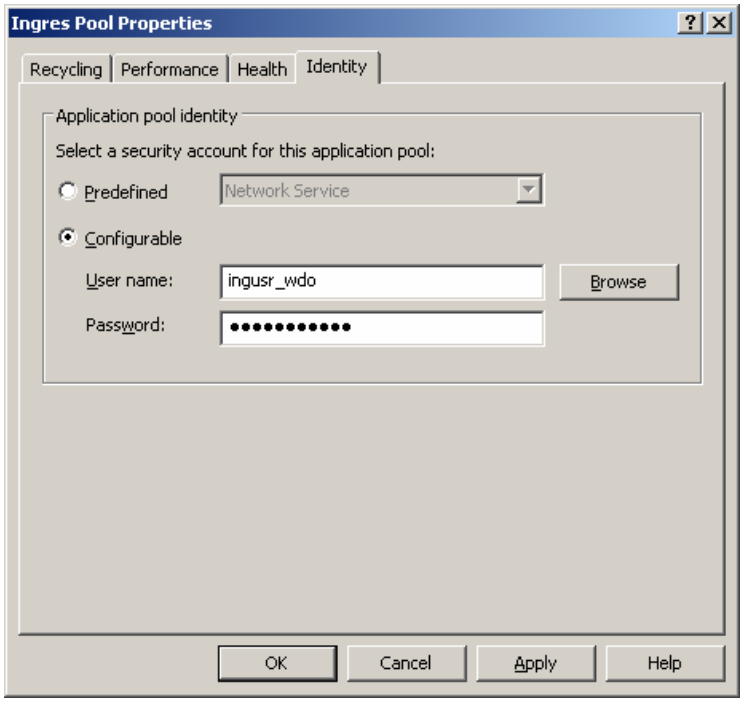
To create a named application pool and assign Web Deployment Option user to identity

1. Start Internet Information Services (IIS) Manager from the Administrative Tools folder.
2. Right click on Application Pools and select New, Application Pool, as shown here:



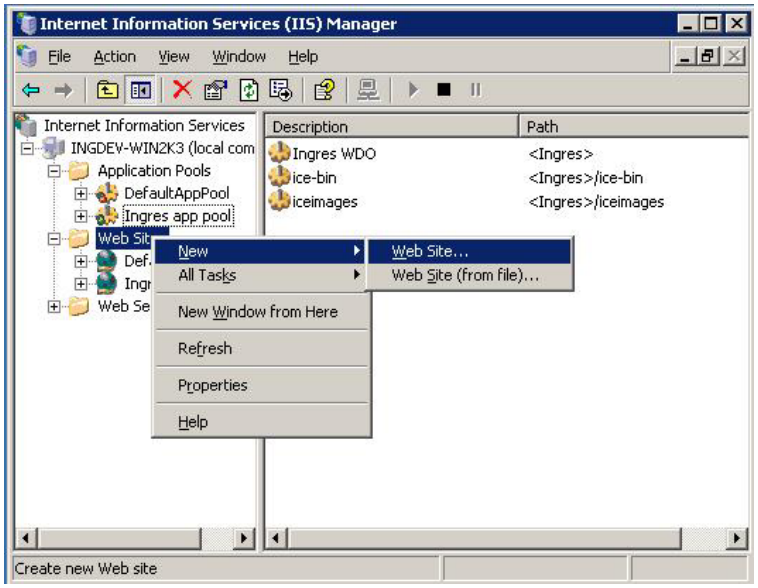
3. Enter a name for this pool (which you will use later) and accept the "Use default settings for new application pool."

Application pool identity property uses the identity of the operating sytem user created previously.

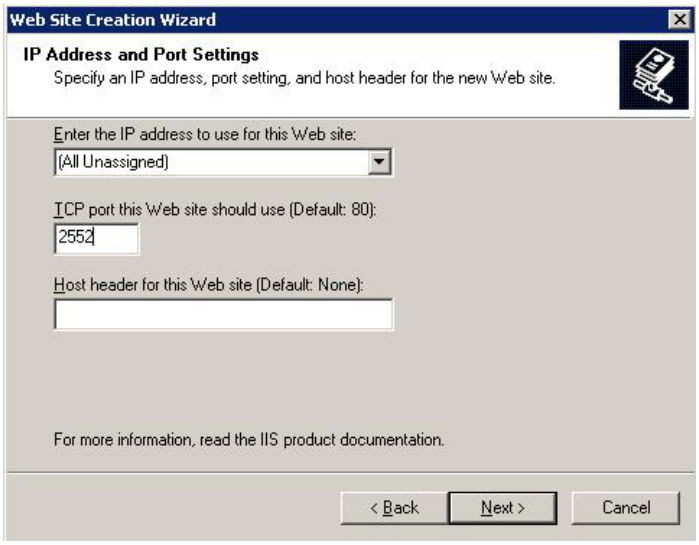


To create a new Web site

1. Right click on Web Sites, and select New, Web Site, as shown here:

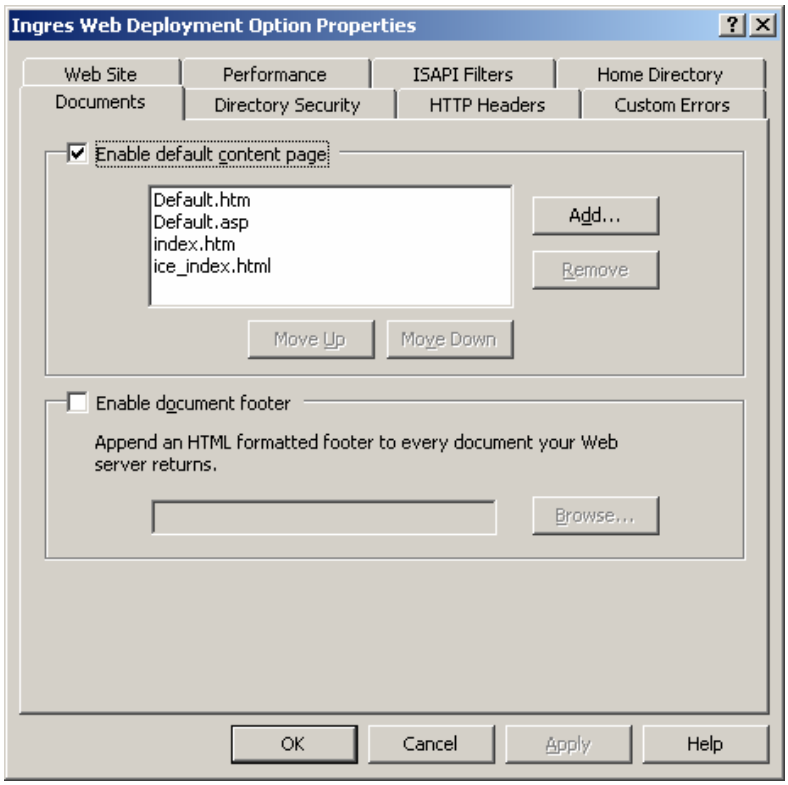


2. Follow the instructions displayed by the Web Site Wizard. Choose a port other than 80. For example:



To add ice_index.html to Documents property for the web site

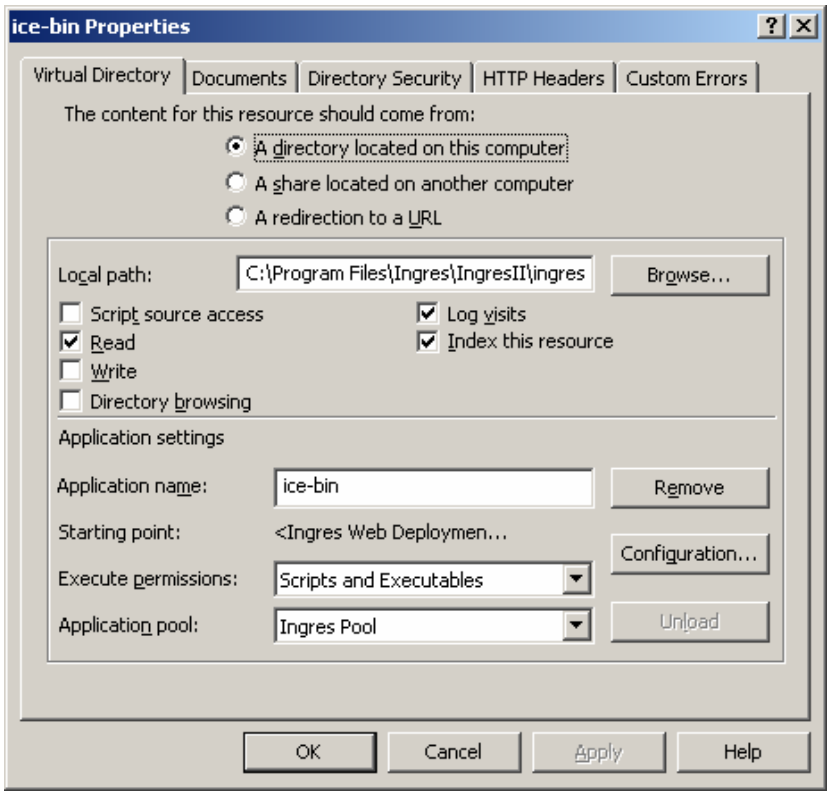
Update the default documents properties to include ice_index.html. For example:



To add virtual directories

1. Add virtual directories, as described in Virtual Directories (IIS).

   When configuring the ice-bin directory, set Executable permissions to Scripts and Executables, and set Application pool to the name you entered previously.



To create new Web services extensions

1. Right click Web Service Extensions and select Add a new Web service extension. Enter a name for the extension. Add required files, namely **oiice.dll** and **oiice.exe**. Check the box "Set extension status to allowed."
2. Reboot the machine.

Virtual Directories (IIS)

To add virtual directories, follow these steps:

1. Click Start on the Windows taskbar, and then choose Settings, Control Panel, Administrative Tools, Internet Services Manager.

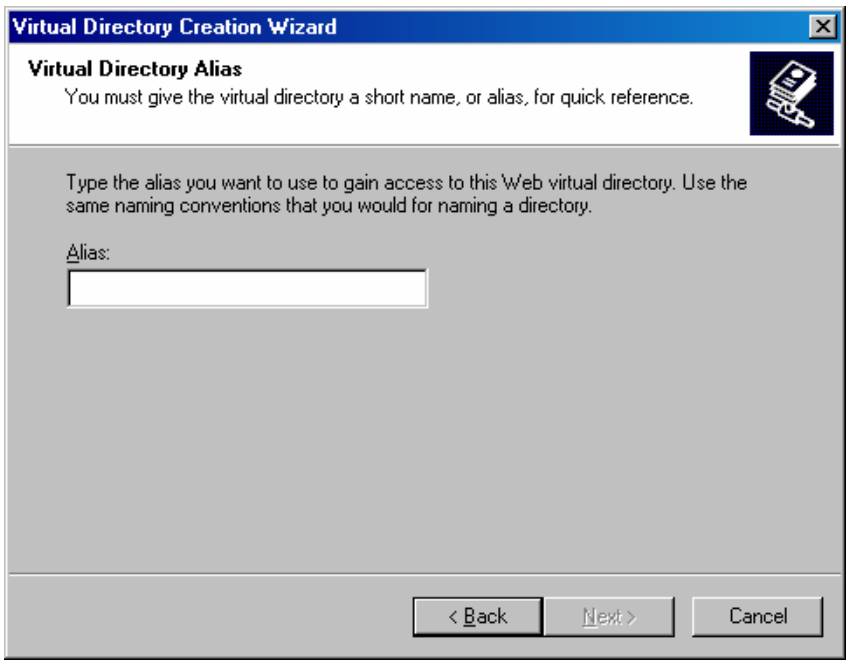
   The Internet Information Services window appears.

2. Expand the branch of the server you want to configure.

3. Right-click the Default Web Site branch.
4. Choose New, Virtual Directory.

   The Welcome to the Virtual Directory Creation Wizard window appears.
5. Click Next.

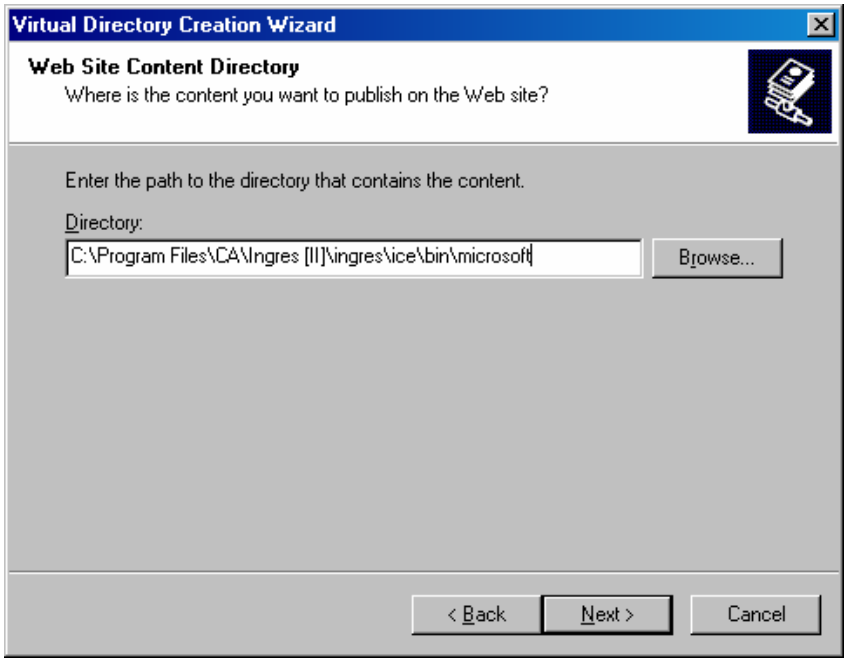
   The Virtual Directory Alias dialog appears:



6. In the Alias field, enter **ice-bin** and click Next.

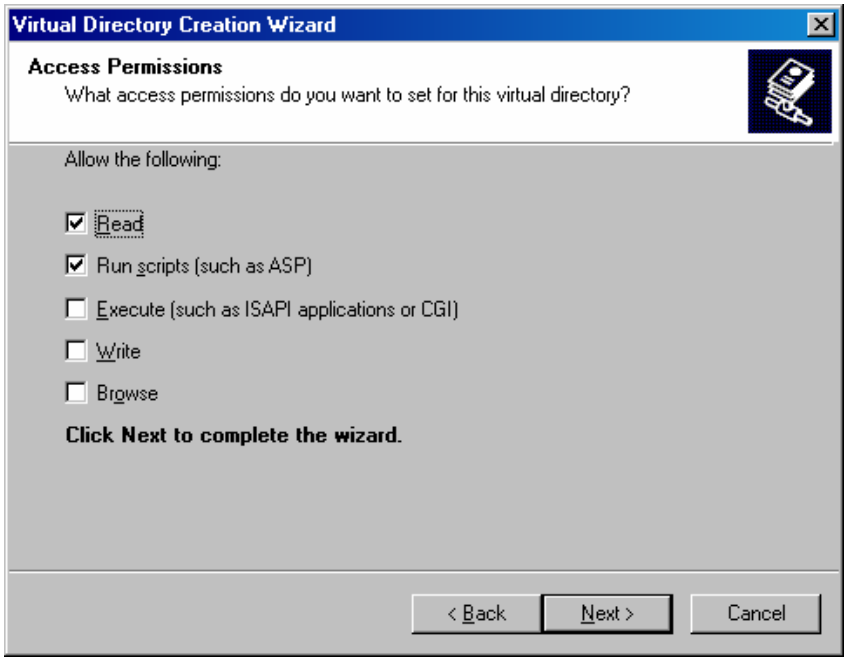
   The Web Site Content Directory dialog appears.

7. In the Directory field, enter the name of the directory where the server extension resides. For example:



8. Click Next.

   The Access Permissions dialog appears:



9. Keep the default values, and then select the Execute check box.
10. Click Next.

11. Click Finish.

    These directories are added to the list under the Default Web Site.

12. Repeat steps 3–11, creating the following directory for the /iceimages alias:

    drive:\<*II_SYSTEM*>**ingres\ice\images**

    Note: Do not select the Execute check box when creating this directory.

ICE File Type (IIS)

The Web path name indicates the ICE file type.

Using Your Web Server as a Windows Service

If you are using your web server as a service, you should restart Windows to ensure that the Services manager incorporates all settings.

Apache Web Server

To configure the Apache Web Server, edit the **httpd.conf** file in the Apache **conf/** directory.

Note: After you update the configuration file you must restart the Apache Web Server.

Tip: The Apache Web Server supports an **include** directive that lets multiple configuration files be included in the server configuration. You can keep configuration information either in the **httpd.conf** file or in one that is included by it. This feature allows web server administrators to store changes for particular services in individual **conf** files.

An example include conf file for the ICE Server is on the distribution media in <*II_SYSTEM*>/ingres/ice/bin/apache/**ice.conf**. This file uses the virtual server capability of the Apache Web Server, which listens on a different port. You can use this file as an example to set up your Apache Web Server as quickly as possible.

Environment (Apache)

Windows

See Environment (IIS) in this chapter.

UNIX

The following environment variables are required for the Apache Web Server to connect to the ICE Server:

- II_SYSTEM
- LD_LIBRARY_PATH

You must make sure that the process that starts the web server has the usual Ingres environment variables set either for the process or in the server startup script.

If you intend to use the CGI interface, in addition to having these environment variables available to the Apache Web Server process, you must insert the following directive in the **httpd.conf** file to explicitly instruct Apache to pass them on:

```
PassEnv II_SYSTEM LD_LIBRARY_PATH
```

Virtual Directories (Apache)

To add virtual directories for the Apache Web Server, edit the **httpd.conf** file.

You must add virtual directories for ice-bin and iceimages using the **Alias** directive (there is an Alias section in the configuration file). For example:

Windows

```
Alias /ice-bin/ “C:/IngresII/ingres/ice/bin”
Alias /iceimages/ "C:/IngresII/ingres/ice/images"
```

UNIX

```
Alias /ice-bin/ "/IngresII/ingres/ice/bin"
Alias /iceimages/ "/IngresII/ingres/ice/images"
```

ICE File Type (Apache)

You must register the ICE module with the Apache Web Server by loading the ICE module and setting a handler for a particular location. Permissions are assigned using the **Directory** element. The ICE extension is loaded using the **LoadModule** directive. The handler is specified in a **Location** element as illustrated below.

Note: Modules in the form of shared libraries or dynamic link libraries should be identified by absolute path name, for example:

```
<II_SYSTEM>/ingres/ice/bin/apache
```

where <*II_SYSTEM*> represents the value of the environment variable II_SYSTEM for your installation.

Windows

```
# Add the ICE extension module.
LoadModule ice_module <II_SYSTEM>/ingres/ice/bin/apache/oiice.dll

#Locations
<Location /ice-bin>
    SetHandler ice-ext
</Location>

<Directory <II_SYSTEM>/ingres/ice/bin/apache>
    AllowOverride None
    Options None
</Directory>
```

UNIX

```
# Add the ICE extension module.
LoadModule ice_module <II_SYSTEM>/ingres/ice/bin/apache/oiice.1.so

# Enable the apapi module:
AddModule apapi.c

# Locations
<Location /ice-bin>
    SetHandler ice-ext
</Location>

<Directory <II_SYSTEM>/ingres/ice/bin/apache>
    AllowOverride None
    Options None
</Directory>

# Environment variables
PassEnv II_SYSTEM LD_LIBRARY_PATH
```

Automatic Recognition of File Type

The Apache Web Server lets you create "virtual servers." A virtual server can process pages automatically with the ICE Server. An example of how to do this is presented in the **ice.conf** file, located in the following directory:

Windows

%II_SYSTEM%\ingres\ice\bin\apache

UNIX

$II_SYSTEM/ingres/ice/bin/apache

Using Your Web Server as a Windows Service

Windows

If you are using your web server as a service, you should restart Windows to ensure that the Services manager incorporates all settings.

Setting Up Your ICE Server

The ICE Server and its supporting library files are installed in the following directory:

Windows

%II_SYSTEM%\ingres\ice\bin

UNIX

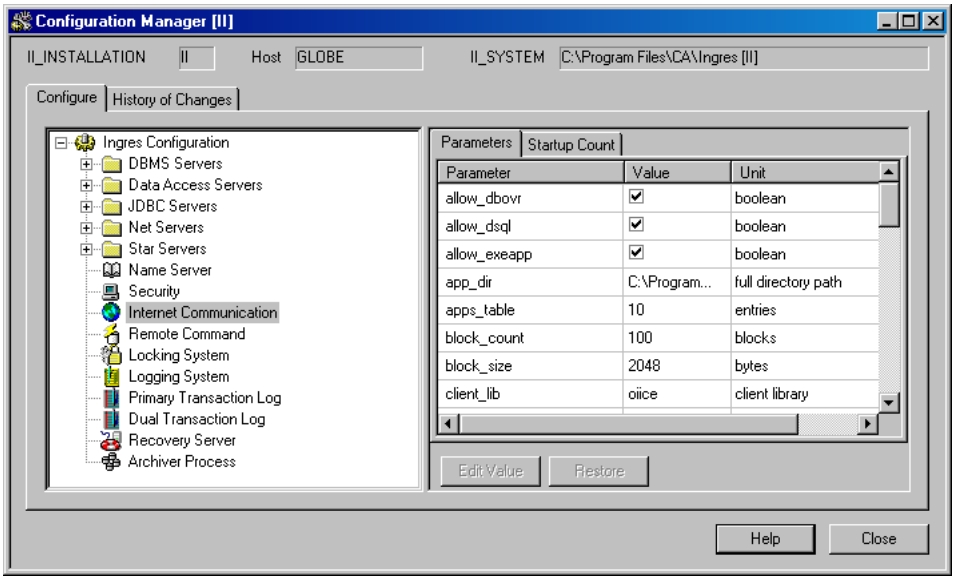
$II_SYSTEM/ingres/ice/bin

It is not necessary to configure the ICE Server files. You can, however, set up your own default system parameters for Internet Communication in the Configuration Manager.

To access Configuration Manager, do the following:

- On Windows, click Start on the taskbar, and then choose Programs, Ingres, Configuration Manager.
- On UNIX, enter **vcbf** at the command line. (For information on the vcbf command, see the *Command Reference Guide*.)

The Configuration Manager is invoked, as shown next. Select the Internet Communication component in the left pane of the window. The configurable parameters for the ICE Server are displayed in the right pane.



For information on the individual parameters, see the online help by pressing F1 while the Internet Communication component and the Parameters tab are selected. Also see the Using the Configuration Manager topic in the Procedures section.

Web Server Document Directory

If you did not set the Web Server Document Root Directory at Web Deployment Option install time, you must set this parameter now. A valid value is essential for communication with your Ingres database. The parameter name is **html_home**, and the value is the full path to the primary HTML document directory of your web server.

Chapter 3: Understanding the Web Deployment Option

This chapter provides an overview of the Web Deployment Option, including its powerful database web server and the facilities it provides for managing a web/client/server environment.

Overview

The Web Deployment Option provides a host of features that allow the application developer to create dynamic web-based applications that are independent of the web user's browser and the data sources with which they communicate. Essentially, applications can be developed with only HTML code.

The Web Deployment Option also offers solutions to important problems currently encountered by web application developers today. These include the flexibility to deliver solutions through dynamic HTML pages, while protecting and managing access to their enterprise's data.

The main features of the Web Deployment Option include:

- Session capability

  Web access is usually stateless, starting new connections with the request of each HTML page. When developing applications for the Web, it is desirable that state information is maintained between pages. Using session management, it is possible to maintain a session context by using a unique cookie identifier. A cookie is the unique data that identifies a remote user.

- Security management

  Since the volume of users in a web environment can become very large, creating a system or database account for each potential client on the operating system is unrealistic. Therefore, a logical user or "web user" is created for the Web Deployment Option, enforcing user authentication before a session to the Web Deployment Option is opened. User rights can be established in the system and access to HTML pages can be assigned.

- Transaction processing

  The Web Deployment Option allows HTML developers to open, commit, and roll back transactions through the Web Deployment Option macro language. A named transaction allows the handling of query operations performed by a user (for example, browsing or selecting items from a list) over many HTML pages, and then the committal of the transaction only when the user is finished.

- Macro language

  The Web Deployment Option provides an extensive macro language that allows the web author to embed Web Deployment Option macros in HTML documents to execute SQL statements and automatically format the result sets. In addition, a connection protocol is used that authenticates a user before establishing a Web Deployment Option session.

- Document caching

  When dealing with web documents, it is possible for a single HTML page to reference many different facets; each facet may vary in size and complexity. Using the ICE Server, there can be many facets stored within the database and extracted each time they are needed. Adding a document to the cache reduces the number of database accesses.

- Administration and management tools

  Visual DBA enables you to administer your Web Deployment Option system, including security, business unit, and server information. You can also monitor various entities, such as connected and active users, cached pages, HTTP and database connections, transactions, and cursors.

Users

The users of Web Deployment Option fall into three major categories including the roles of administrator, web author, or web user.

These roles are defined as follows:

- Web Deployment Option administrator

  There are several types of administrators in the Web Deployment Option, including:

  – Privileged user

    Has all permissions and privileges on the ICE Server. This user is equivalent to the Ingres administrator (installation owner) or the root user on UNIX.

  – Server administrator

    Manages session groups, locations, and server variables.

- Security administrator

  Manages all security components, including web users, database users, roles, profiles, and database connections.

- Business unit manager

  Creates and manages a business unit—which entails defining the list of locations available through this business unit, declaring authorizations, and managing the unit's backup.

- Monitor

  Views and keeps track of Web Deployment Option system information.

- Web author

  Someone who is responsible for analyzing user and enterprise requirements, as well as designing and implementing Web Deployment Option application programs to meet those requirements. The web author is also responsible for validating that the application programs meet those requirements, and for handing them over to the Web Deployment Option business unit manager for installation.

  Web authors can also be given authorization to manage security, projects, HTML pages, and multimedia documents in the system.

- Web user

  Any user declared in the Web Deployment Option repository and who is permitted to request documents from the ICE Server. This includes administrators, developers, and end-users of applications.

  Note: A default database user alias is associated with each web user, which determines the data source the user accesses. Through database connections, a web user can also be associated with other database users and data sources.

Architecture

ICE Server architecture is based on the Ingres DBMS Server architecture and allows multiple users access to databases through connections to one or more ICE Server processes. The ICE Server is also a multi-threaded daemon process that generates dynamic page content and manages resource pooling with data sources (such as Ingres).

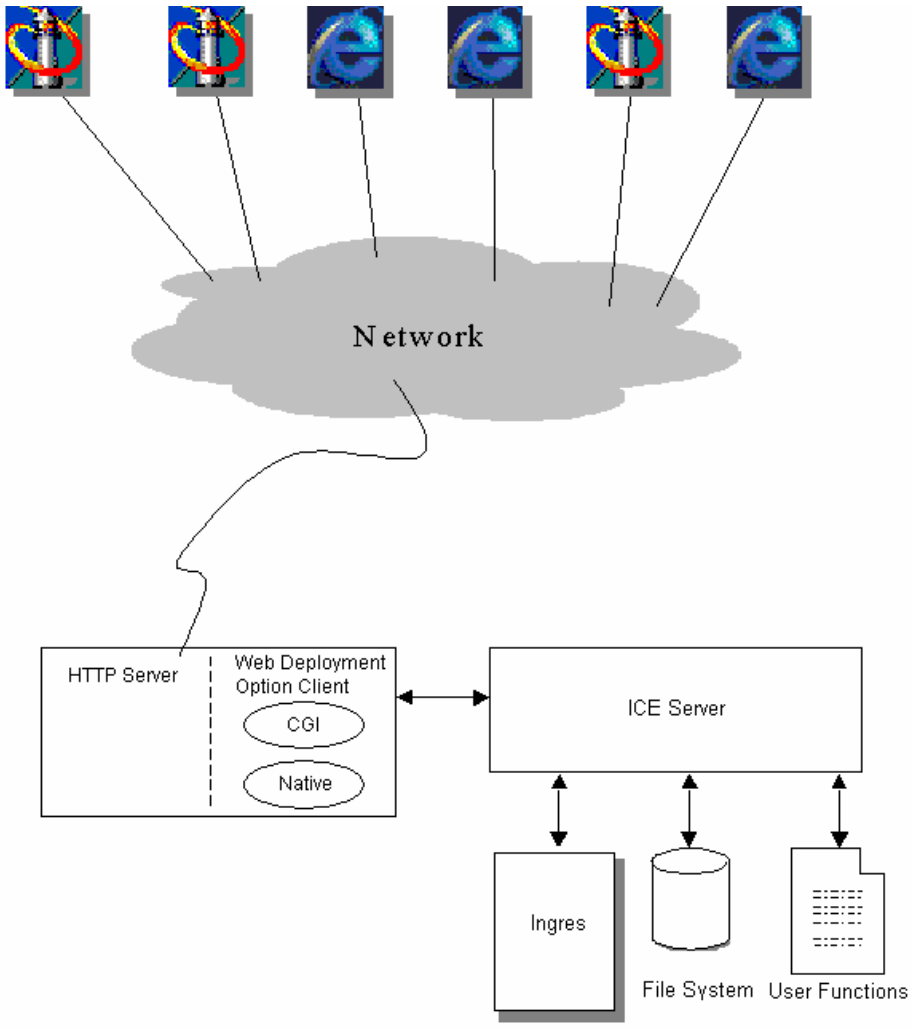
The ICE Server architecture provides a separate application layer that can pass information to any web browser. The browser is then responsible for interpreting the information for the web user. Web Deployment Option is designed to enforce security in the web environment, allow scalability for more users, integrate with emerging products, and leverage existing business logic.

Web Site Components

The primary components of a Web Deployment Option web site are:

- Browser
- HTTP server and Web Deployment Option client
- ICE Server
- Information systems

The figure below illustrates a Web Deployment Option web site:



Web Browser

A web browser allows a user to interact with a web application. It also allows any web user with the proper privileges to configure and manage the Web Deployment Option. The Web Deployment Option allows the development of web applications with essentially only HTML code. Therefore, the browser is independent of the web application.

Web Deployment Option Client

The web server executes programs in response to browser requests. CGI is the standard protocol that defines the way the server executes the programs. The communication between the HTTP server and the ICE Server can occur through a CGI external executable or an internal native HTTP server extension. CGI and native drivers act as "communications pipes" sending requests to and receiving results from the ICE Server.

CGI Extensions

The Web Deployment Option is supplied as a CGI-compliant executable file and should support any web server that also supports CGI. In addition, the Web Deployment Option client is also supplied as a native extension to some of the more common HTTP servers.

Native Extensions

There are a number of limitations to the CGI approach, and for this reason, web server vendors have defined their own methods for extending the capabilities of their products. The Web Deployment Option is also supplied in a form compatible with three of the most widely supported web server native API driver interfaces, including:

- ISAPI—Microsoft Internet Server API
- Apache API—Apache Server API

Windows

In Windows environments, Web Deployment Option components written to these interfaces are supplied in the form of a Windows Dynamic Link Library (DLL).

UNIX

In UNIX, Web Deployment Option components written to these interfaces are supplied in the form of a UNIX shared library.

These interfaces are otherwise functionally identical to the CGI executable. They provide performance benefits and allow HTML developers freedom to develop standard Web Deployment Option web applications or pages without providing a separate interface for each environment.

When you create a Web Deployment Option application, you need to refer to the CGI client or native extension in your HTML. Typically, this will be in the Uniform Resource Identifier (URI) you supply in the FORM ACTION tag. The name of the executable to which you refer depends on the web server interface you have chosen.

The following table lists the possible interfaces for the supported web servers:

Web Server Interface	Windows File Name	UNIX File Name
CGI application	oiice.exe	oiice
Microsoft ISAPI library	oiice.dll	Not available
Apache	oiice.dll	oiice.1

ICE Server

The multi-threading ICE Server communicates with the HTTP server either by using a CGI executable or a server extension. It communicates with the data source passing requests to the source and returning the results to the browser.

The ICE Server manages all connections, session information, and security for those areas of the web site under its control. It maintains a list of users, their privileges, and roles (used for simplifying the maintenance of the privileges for a group of users).

The documents that can be viewed by users fall into a three-layer hierarchy: the documents themselves, business units (which are a collection of active pages and facets), and session groups (which are a logical grouping of business units).

In addition to the powerful Web Deployment Option macro language, support for running Ingres reports, database procedures, and user written applications, the ICE Server offers built-in functions, a C API, and user-defined functions.

The ICE Server maintains all of this information in a repository, which is maintained by the ICE Server administrator.

Information Systems

The Web Deployment Option supports Ingres databases and all Enterprise Access products (gateways), which provide access to other data sources such as Microsoft SQL Server, Sybase, Informix, Oracle, CA-IDMS, CA-Datacom, among others.

Chapter 4: Managing the Web Deployment Option

This chapter describes the Web Deployment Option objects that you can manage, including security, business unit, and server objects. The following topics are discussed:

- Accessing Web Deployment Option information
- Managing database users, database connections, roles, profiles, and Web users
- Managing user and role access for a business unit, pages, facets, and associations to locations
- Managing session groups, locations, and server variables
- Monitoring Web Deployment Option information

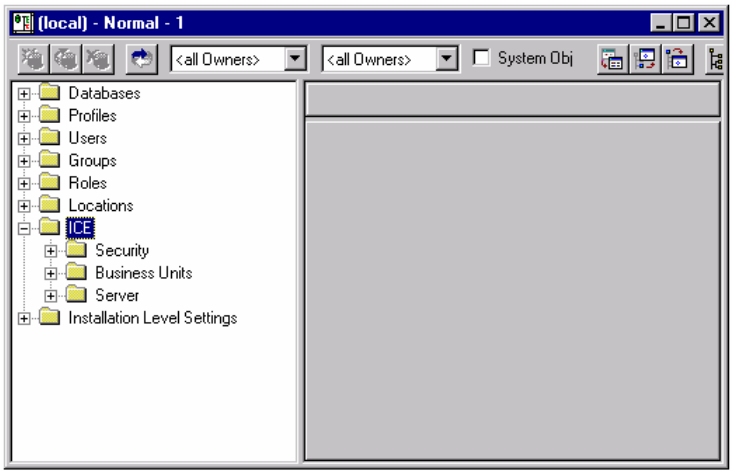
The server administrator, security administrator, and business unit administrator will be managing the various objects in the Web Deployment Option system. A user with the monitor privilege can monitor the Web Deployment Option system.

Accessing Web Deployment Option Information

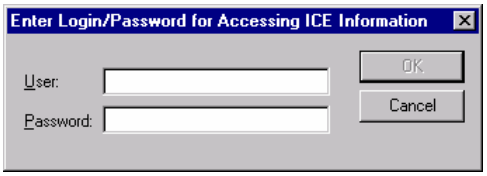
Visual DBA allows you to view and manage your Web Deployment Option objects. On Windows, you start Visual DBA by clicking Start on the taskbar, and then choosing Programs, Ingres, Visual DBA. On UNIX, enter vdba at the command prompt.

To open a Database Object Manager window, expand the Nodes branch and select a virtual node. Then, click the Connect DOM toolbar button.

To access the Web Deployment Option objects on your installation, you expand the ICE root branch in the Database Object Manager. This reveals the following sub-branches, which represent Web Deployment Option object categories:



If you expand one of the sub-branches further, you either see other object categories or you are prompted with the following dialog:



Enter the name and password associated with the Web Deployment Option privileged user. This user is the installation owner set up during installation (default ID is ingres), and must be defined as an operating system user. When you are finished, click OK.

Note: The Web Deployment Option privileged user is defined in the config.dat file and can be changed to another Ingres administrator using the Configuration Manager utility.

The objects appear under the branch, provided you have access privileges to the information. The section Managing Security describes how Web users are granted privileges to resources once they are defined.

Managing Security

The Web Deployment Option provides a variety of mechanisms for managing Web resources and granting access to them. It also allows for the security of the data sources in the installation.

If you are a Web Deployment Option privileged user or a security administrator, you can create the following:

- Web users
- Database users
- Database connections
- Roles
- Profiles

Web Users

A W*eb user* is any user declared in the Web Deployment Option repository and is permitted to request documents from the ICE Server. The user may be an administrator, developer, or end user of an application. The security administrator manages Web users and their associations to roles and database connections.

There are two different ways in which Web users can be created. A security administrator can manually create a Web user definition through Visual DBA. Web users can also be automatically defined through a series of "auto declaration" statements defined in a Web page. A new Web user is instructed by an application to enter a user name and password—resulting in the user being auto-declared through a profile created by the security administrator that defines the user type and attributes.

Working with Web User Objects

In Visual DBA, Web users are implemented using *Web user objects*. A Web user object specifies the user's name, password, the type of user, and several other attributes.

Using the Web Users branch in the Database Object Manager window, you can:

- Create and alter Web user objects
- View existing Web user objects, including the detailed properties of individual objects
- Drop Web user objects

- Associate role objects with Web user objects
- Associate database connection objects with Web user objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Web User
- Altering a Web User
- Viewing Object Properties
- Associating a Role with a Web User
- Associating a Database Connection with a Web User
- Dropping Objects

How Web Users Are Used

Once a Web user is created, you can associate several types of objects with it. The associations define which data sources the user can access and what access permissions the user has to Web resources. The following types of objects can be associated with a Web user:

- Role

  A role can be associated with a Web user and privileges associated with it for a business unit.

- Database connection

  One or more database connections can also be associated with a Web user to identify which database connections the user can use.

- Business unit

  A business unit can also be associated with a Web user, which gives the user access to the pages and facets in the business unit.

- Pages and facets

  To provide a finer granularity of security based on the page or facet the user is trying to access, individual pages and facets can be associated with a Web user.

For details on how to create associations between these types of objects and a Web user, see Role Access Definitions in this chapter.

Database Users

When accessing the information system, the Web Deployment Option uses a *database user* definition. A database user is a user alias that maps to an actual user (such as "ingres"), for which a database administrator grants permissions.

One default database user can be used by a large number of Web users (or profiles) to access a data source. This allows the number of users known to the data source to be minimized, while the number of Web users can be quite large.

The database user is aliased by the Web Deployment Option to an Ingres database user. The user can be specified in a Web Deployment Option macro for public access—although this is not recommended. (This is provided for Web Deployment Option (formerly Ingres/ICE) Version 2.0 compatibility, however the use of the features in versions 2.5 and higher is preferred.)

Working with Database User Objects

In Visual DBA, Web Deployment Option database users are defined using *database user objects*. A database user object specifies the user's alias, name, password, and an optional comment.

Using the Database Users branch in the Database Object Manager window, you can:

- Create and alter database user objects
- View existing database user objects, including the detailed properties of individual objects
- Drop database user objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Database User
- Altering a Database User
- Viewing Object Properties
- Dropping Objects

How Database Users Are Used

Once a database user is created, you can assign it as the default database user when creating a new Web user or profile. This instructs the Web Deployment Option to use the user name and password defined in the database user definition by default when connecting to a data source. This is how the Web Deployment Option administrator arranges for multiple Web users to access the data source using one user name for that source.

One or more database connections can also be associated with a database user. For more information, see Database Connections in this chapter.

Database Connections

A *database connection* allows the Web author to use an alias for a database and a database user. A Web user is associated with a database user for the purpose of convenience and to abstract internal names.

Working with Database Connections

In Visual DBA, database connections are defined using *database connection objects*. A database connection object specifies the database connection's name, virtual node, database, database user, and an optional comment.

Using the Database Connections branch (beneath the ICE Security sub-branch) in the Database Object Manager window, you can:

- Create and alter database connection objects
- View existing database connection objects, including the detailed properties of individual objects
- Drop database connection objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Database Connection
- Altering a Database Connection
- Viewing Object Properties
- Dropping Objects

How Database Connections Are Used

Database connections are used to control access to the database and to make the database location and user transparent, thus concealing the true names of the database and the database user from the Web. In addition to providing abstraction, database connections allow for simpler code since a developer needs only to reference a database connection in a page.

For each Web user, you can define the database connections the user will be able to use. A Web user is defined with a default database connection with which to use if none is specified.

Roles

You can streamline the user authorization process using *roles*. A Web Deployment Option role is a logical entity that allows a security administrator to give authority to a set of Web users.

You can define a set of permissions for the role on the business unit, page, or facet level.

Working with Role Objects

In Visual DBA, Web Deployment Option roles are defined using *role objects*. A role object specifies the role's name and an optional comment.

Using the Roles branch (under the ICE Security sub-branch) in the Database Object Manager window, you can:

- Create and alter role objects
- View existing role objects, including the detailed properties of individual objects
- Drop role objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Role
- Altering a Role
- Viewing Object Properties
- Dropping Objects

How Role Objects Are Used

Once a role is created, a role can be associated with many existing Web users instead of defining and updating the security privileges individually for each user. A set of predefined permissions can be defined for a role through a *role access definition* for a particular business unit, page, or facet.

Business units define which sort of permissions are to be allowed on a per-role or per-user basis. A user associated with a role will therefore automatically be assigned the permissions of that role. A user can be granted other permissions individually.

For more information, see Role Access Definitions in this chapter.

Note: When defining a role, the security administrator typically works with the Web author, so that they can agree on the identifier and the permissions the role has for specific applications.

Profiles

A *profile* is used to set up a default level of access to a business unit for a Web user. It does this by associating the Web user with a role. The business unit then defines the level of access for that role. Profiles allow for simpler user management—predefining Web user definitions.

Working with Profile Objects

In Visual DBA, profiles are implemented using *profile objects*. A profile object specifies the profile's name, default database user, type of user, and time-out duration between requests from the server.

Using the Profiles branch in the Database Object Manager window, you can:

- Create and alter profile objects
- View existing profile objects, including the detailed properties of individual objects
- Drop profile objects
- Associate a role object with a profile object
- Associate a database connection object with a profile object

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Profile
- Altering a Profile

- Viewing Object Properties
- Associating a Role with a Profile
- Associating a Database Connection with a Profile
- Dropping Objects

How Profiles Are Used

Once a profile is defined, a role or database connection can be associated with it. With a role, permissions for a business unit or specific Web resource can be defined. Database connections allow you to control which data sources can be accessed for a profile.

For more information on these objects, see Roles and Database Connections in this chapter.

Managing Server Information

Several entities relate to the ICE Server and provide the ability to group document elements together logically, specify server locations that can be used to store pages and facets, and store server variables that can be referenced in documents.

Session Groups

A *session group* defines a logical group of business units. Each business unit must belong to a session group. No additional security or access permissions are associated with a session group. A session group identifies a unique cookie for a session.

Working with Session Groups

In Visual DBA, session groups are defined using *session group objects*. A session group object specifies the name of the session group.

Using the Session Groups branch (beneath the ICE Security sub-branch) in the Database Object Manager window, you can:

- Create and alter session group objects
- View existing session group objects, including the detailed properties of individual objects
- Drop session group objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Session Group
- Viewing Object Properties
- Dropping Objects

How Session Groups Are Used

A session group specifies a set of Web Deployment Option documents. It is used in the generation of a Web Deployment Option session cookie. Using a session group enables a client to open multiple session groups on the same client with multiple instances of the browser.

The Web Deployment Option uses sessions to manage connections. For a secured site, each user has to identify himself by opening a connection on the ICE Server. This named connection allows the server to keep track of multiple session contexts from a single client running different Web applications concurrently. When a user logs in to the Web Deployment Option, a cookie is assigned. The session group then becomes valid.

To work with session groups, you need to be a Web Deployment Option privileged user or server administrator. The privileged user can create a location for use by a business unit manager or owner for Ingres II 2.0 compatibility access. For more information, see Managing Security in this chapter.

After you have created a session group object, you can use the new session group to reference a page mapped to a location through a Web Deployment Option address. For the syntax of this address, see "Appendix B: HTML Primer."

Note: Login and auto-declare pages are public and should *not* specify session groups whereas secure pages should.

Locations

The ICE Server references each active web resource, whether it is a page or facet, using a *location*. There are two main types of locations: an HTTP-visible location, and a Web Deployment Option location, which is invisible to HTTP. A location that is controlled by the Web Deployment Option is invisible to the HTTP server.

In addition, locations can be marked public (anyone can view their contents), or not public, in which case only those users who have logged in can view their contents.

Working with Locations

In Visual DBA, locations are defined using *location objects*. A location object specifies the location's name, path, location type, extension priorities, a comment, and whether it is public.

Using the Locations sub-branch under the ICE Server branch in the Database Object Manager window, you can:

- Create and alter Web Deployment Option location objects
- View existing Web Deployment Option location objects, including the detailed properties of individual objects
- Drop Web Deployment Option location objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Web Deployment Option Location
- Altering a Web Deployment Option Location
- Viewing Object Properties
- Dropping Objects

To work with Web Deployment Option locations, you need to be a Web Deployment Option privileged user or a server administrator. The privileged user can grant the privilege to a Web user by associating the administrator profile, through profile or Web user definitions. For more information, see Managing Security in this chapter.

How Locations Are Used

Locations allow you to group related resources into specific directories and abstract the file system through aliasing. Files held in these directories can be accessed by the Web Deployment Option or by the HTTP server.

After you have set up the path and mapped it to a location by creating a location object, you can use the new location to reference a page mapped to a location through a Web Deployment Option address. For the syntax of this address, see "Appendix B: HTML Primer."

ICE Server Variables

A *server variable* is a reference value associated with the ICE Server and is held internally by the ICE Server.

Working with ICE Server Variables

In Visual DBA, server variables are defined using *server variable objects*. A server variable object specifies the variable's name and value.

Using the Server Variables branch in the Database Object Manager window, you can:

- Create and alter server variable objects
- View existing server variable objects, including the detailed properties of individual objects
- Drop server variable objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Server Variable
- Altering a Server Variable
- Viewing Object Properties
- Dropping Objects

To work with ICE Server variables, you need to be a Web Deployment Option privileged user or a server administrator. The privileged user can create a server variable that is accessible through the document content, or a developer may allow a server variable to be set from a Web page. For more information, see Managing Security in this chapter.

How Server Variable Objects Are Used

Server variables are used by the Web Deployment Option to hold static information concerned with the installation (such as contact information, this month's include file name, browser type, or possibly the date). The variable is loaded when the ICE Server starts and persists as long as the repository database is not deleted. A reference to a variable is replaced by the actual text when Web Deployment Option parses the file.

Session variables are used to pass values between pages, and page variables are used in a page. All server, session, and page variables share the same name space.

Important! Since all server, session, and HTML variables share the same name space, you should not create a server variable with the same name as a session or page variable. We recommend that your site adopt a convention similar to that in the Web Deployment Option Plays tutorial, discussed in the chapter "Creating Web Applications: An Example."

Managing Business Units

The Web Deployment Option uses business units to manage Web resources, including pages and facets, and to simplify the management of security associated with them.

Business Units

A *business unit* is a group of pages and facets that provide a similar function. For a business unit, a set of locations is defined, which is used to reference and hold the page and facet files. Defining business units enables simpler document management.

Each file must belong to a business unit to reference it in URIs or your HTML code. You can think of the business unit as being like an operating system directory (that contains files like pages, facets, or both).

Working with Business Units

In Visual DBA, business units are defined using *business unit objects*. A business unit object specifies the name of the business unit.

Using the Business Units branch in the Database Object Manager window, you can:

- Create and alter business unit objects
- View existing business unit objects, including the detailed properties of individual objects
- Drop business unit objects
- Create and alter role access definitions for a business unit
- Create and alter user access definitions for a business unit
- Associate page or facet objects with business unit objects
- Back up business unit objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Business Unit
- Viewing Object Properties
- Dropping Objects
- Creating a Role Access Definition for a Business Unit
- Altering a Role Access Definition for a Business Unit

- Creating a User Access Definition for a Business Unit
- Altering a User Access Definition for a Business Unit
- Associating a Page with a Business Unit
- Associating a Facet with a Business Unit
- Backing Up a Business Unit

Adding Multiple Files to a Business Unit

When first creating a business unit, you may want to register many files to a business unit simultaneously. The Web Deployment Option provides a command line utility, regdocs, to do this and a companion utility, deregdocs, to deregister the documents from a business unit. Using these commands makes it unnecessary to add or remove files individually using Visual DBA.

For the syntax of the regdocs and deregdocs commands, see the *Command Reference Guide*.

How Business Units Are Used

Once a business unit is identified, the business unit administrator can apply security to the business unit as a whole, or to its individual pages or facets. Roles and users can be associated with business units, pages, or facets.

For example, a company might have an accounting business unit to identify the accounting department Web resources as a whole, and a payroll business unit to identify the payroll department Web resources as a whole. To define these business units, the business unit administrator would create them, and then add all the pages and facets for the associated department. The business units could then be easily maintained by adding and dropping resources as they are needed or not needed by the departments.

After you have set up the path for a page file, and mapped it to a location by creating a location object, you can use the new location to reference the page through a Web Deployment Option address. For the syntax of this address, see "Appendix B: HTML Primer."

Note: Pages or facets that are kept as files in the file system can be included in more than one business unit. This is not true for objects that are held in the database.

Documents, Pages, and Facets

A *document* is comprised of *pages* and their related objects, called *facets*. Pages and facets are managed by Web Deployment Option to produce the document the end user sees in a browser. The pages and facets can be stored directly on the file system or in Ingres.

A page is a text file containing Web application language statements (for instance, HTML statements) and copy. A facet is any other type of object that is used on a page (such as an image, style sheet, audio, or multimedia object).

Tip: The Web Deployment Option processes pages, not facets. Therefore, if a style sheet needs to be parsed, it should be treated as a page rather than as a facet.

The Web author creates page and facet objects for use in an application.

Working with Page and Facet Objects

In Visual DBA, Web Deployment Option pages and facets are defined using *facet objects* and *page objects*. A page or facet object specifies the name of the page or facet, path to the file, storage type, and caching method.

Using the Pages and Facets branches in the Database Object Manager window, you can:

- Create and alter page and facet objects for business units
- Create and alter role access definitions for page objects
- Create and alter user access definitions for facet objects
- View existing page and facet objects, including the detailed properties of individual objects
- Drop page and facet objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Page for a Business Unit
- Altering a Page for a Business Unit
- Creating a Facet for a Business Unit
- Altering a Facet for a Business Unit
- Creating a Role Access Definition for a Page

- Altering a Role Access Definition for a Page
- Creating a User Access Definition for a Page
- Altering a User Access Definition for a Page
- Creating a Role Access Definition for a Facet
- Altering a Role Access Definition for a Facet
- Creating a User Access Definition for a Facet
- Altering a User Access Definition for a Facet
- Viewing Object Properties
- Dropping Objects

How Pages and Facets Are Used

Once created, you can reference page and facet objects from within a page.

After you have created a page or facet, you can reference the new page or facet through a Web Deployment Option address. For the syntax of this address, see "Appendix B: HTML Primer."

Role Access Definitions

Role access definitions can be created that define the permissions that are associated with the role for an entire business unit, or an individual page or facet in a business unit.

Working with Role Access Definitions

In Visual DBA, role access definitions are defined using *role access definition objects*. These definitions specify the role's name and the permissions associated with the role.

Using the Roles branches (under the Business Units Security sub-branch, or the Pages and Facets branches) in the Database Object Manager window, you can:

- Create and alter role access definition objects for business units
- Create and alter role access definition objects for pages and facets
- View existing role access definition objects, including the detailed properties of individual objects
- Drop role access definition objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a Role Access Definition for a Business Unit
- Altering a Role Access Definition for a Business Unit
- Creating a Role Access Definition for a Page or Facet
- Altering a Role Access Definition for a Page or Facet
- Viewing Object Properties
- Dropping Objects

How Role Access Definitions Are Used

For a particular business unit, page, or facet, a role access definition can be created that associates permissions with a role. This provides the flexibility to assign different permissions to a role used in different business units.

The object permissions that are applied to the role for a business unit include the ability to create, read, and execute documents. Those permissions for a facet or page include the ability to create, read, update, or delete.

Web User Access Definitions

Web user access definitions can be created that define the permissions that are associated with a Web user for an entire business unit, or an individual page or facet in a business unit.

Working with Web User Access Definitions

In Visual DBA, user access definitions are defined using *user access definition objects*. These definitions specify the user's name and the permissions associated with the user.

Using the Users branch (under the Business Units Security sub-branch) in the Database Object Manager window, you can:

- Create and alter user access definition objects for business units
- Create and alter user access definition objects for pages and facets
- View existing user access definition objects, including the detailed properties of individual objects
- Drop user access definition objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Creating a User Access Definition for a Business Unit
- Altering a User Access Definition for a Business Unit
- Creating a User Access Definition for a Page or Facet
- Altering a User Access Definition for a Page or Facet
- Viewing Object Properties
- Dropping Objects

How Web User Access Definitions Are Used

For a particular business unit, page, or facet, an access definition can be created for the Web user that associates permissions with the user. These permissions, set directly at the Web user level, override any access definition settings at the role level.

The object permissions that are applied to the Web user for a business unit include the ability to create, read, and execute documents. Those permissions for a page or facet include the ability to create, read, update, or delete.

Associating a Location with a Business Unit

You can establish an association between a location and a business unit in order to map Web resources to an actual physical disk location.

Using the Locations branch (under the Business Units branch) in the Database Object Manager window, you can:

- Create associations between locations and business unit objects
- View existing associations between locations and business unit objects, including the detailed properties of individual objects
- Drop associations between location and business unit objects

The detailed steps for performing these procedures can be found in the Procedures section of the online help. See the following topics:

- Associating a Business Unit with a Location
- Viewing Object Properties
- Dropping Objects

Monitoring Web Deployment Option Information

The Performance Monitor window in Visual DBA can be used to view information on the ICE Server. It provides information on the activity that is occurring on the server at any time. The following entities can be monitored:

- ICE Server properties
- Connected users
- Active users
- Cached files
- HTTP server connections
- Database connections
- Transactions
- Cursors

To access this information, select the ICE Server under the Servers branch. In the right pane, you can view detailed information on various entities. For more information, see the Viewing Performance Information topic in the Procedures section of the online help.

Shutting Down

Before shutting down an Ingres installation that includes an ICE Server, you must first terminate any connections that exist between the HTTP server and the ICE Server.

The Web Deployment Option HTTP server extensions maintain a pool of client connections with the ICE Server. The ICE Server detects these connections as active sessions. To disconnect these sessions, the HTTP server extension should be unloaded. If this is not possible, either the HTTP server can be shut down or the Web Deployment Option can be forcibly shut down before shutting down the remainder of the Ingres installation.

For instructions on how to shut down Ingres, see the *System Administrator Guide*.

Chapter 5: Using the Macro Language

This chapter contains reference information that provides instruction on how to embed Web Deployment Option macros into your XML or HTML documents, including descriptions of the Web Deployment Option XML tags and the macro language keywords that are available.

You can use the Web Deployment Option language to do the following:

- Create template documents that retrieve their content directly from the database
- Store images and other multimedia content in an Ingres database and share them as XML or display them with HTML
- Show the results of multiple queries on a single page

Note: The new XML tag language provides the same functionality as the macro keywords presented later in the chapter. However, by using an XML-aware editor, it is much easier to create documents that do the same thing and are valid Web Deployment Option. The macro language keywords are still supported for backward compatibility reasons.

Web Deployment Option XML Tag Set

The Web Deployment Option XML tag set has been defined to be well-formed and conformant XML. It allows XML editors to be used to create Web Deployment Option macro language document templates in a seamless way. The Web Deployment Option Document Type Description (DTD) file can be used stand-alone or in conjunction with the HTML DTD to create HTML/ICE templates to be used in Web Deployment Option web sites.

Web Deployment Option XML Macro Tag Format

The format of a Web Deployment Option macro tag is as follows:

<i3ce_*tag*[*attributes*]>
<i3ce_*child_tag*[*attributes*]**>**
</i3ce_*child_tag*>
</i3ce_*tag*>

In this syntax representation:

- *tag* is a valid Web Deployment Option XML tag
- *child_tag* is any XML tag that is legal in the context of the parent tag
- *attributes* represents any attributes that are legal in the context of the tag

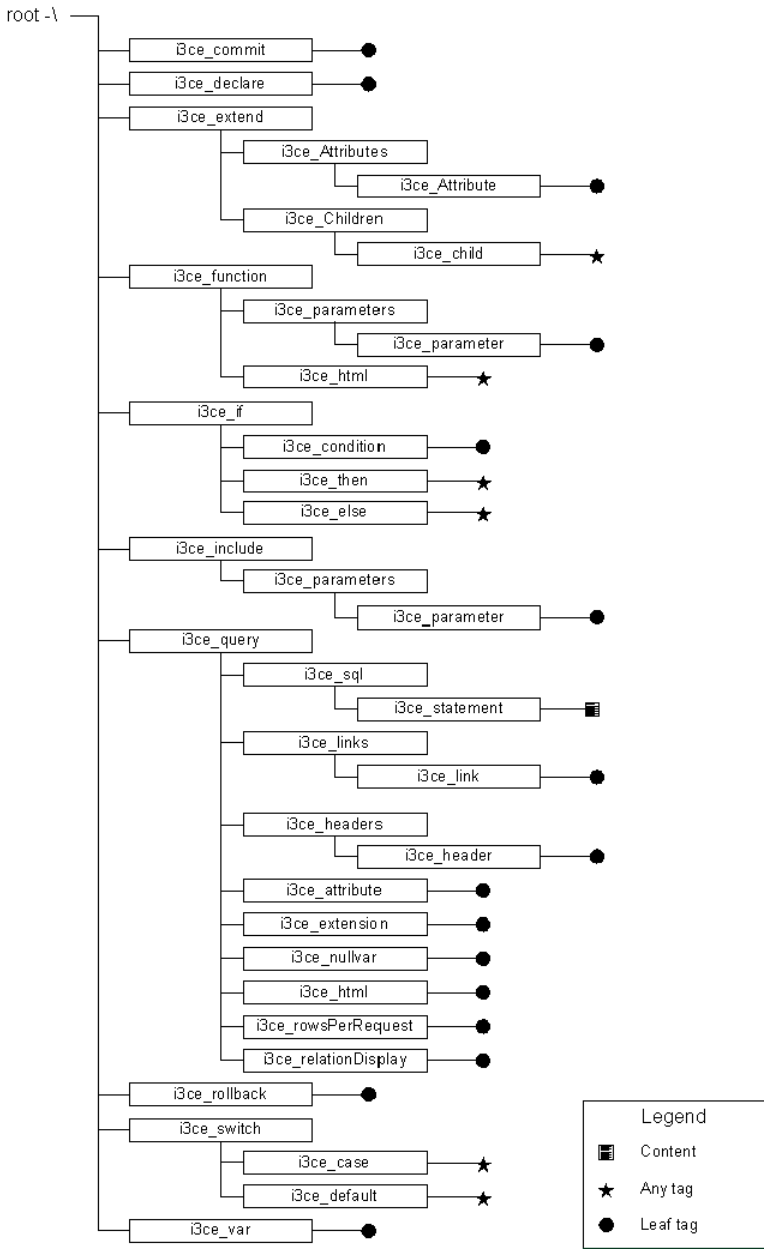
Web Deployment Option Macro Tags

The following Web Deployment Option macro tags are available and are described in more detail in the sections that follow:

- <i3ce_commit>
- <i3ce_declare>
- <i3ce_extend>
- <i3ce_function>
- <i3ce_if>
- <i3ce_include>
- <i3ce_query>
- <i3ce_rollback>
- <i3ce_switch>

Tag Hierarchy

Some Web Deployment Option macro tags can contain children. The tag hierarchy is shown below:



Notes:

- "No children" indicates a leaf tag. This tag takes no content, and is an Empty Tag. Any information that needs to be supplied to the Web Deployment Option at this point is specified in the Tag Attributes.

- "Any tag" is used to refer to any HTML or Web Deployment Option tag that would be legal in the current context. The i3ce_HTML tag uses this, for example, to allow flexibility in completing HTML elements. Since it effectively opens a loophole in the DTD and subverts the normal XML syntax checking, it should be used with caution!

Macro Tags

This section presents the syntax for each Web Deployment Option macro tag.

<i3ce_commit> Tag

Purpose

Commits a previously started transaction.

Syntax

<i3ce_commit i3ce_transaction="*transaction_name***"/>**

Description

The <i3ce_commit> tag commits the named transaction, *transaction_name*. The transaction name must have been defined previously using the i3ce_transaction attribute of the <i3ce_sql> tag.

Example

```
<i3ce_commit i3ce_transaction="myTransaction"/>
```

See Also

i3ce_query, i3ce_rollback

<i3ce_declare> Tag

Purpose

Assigns a value to a named variable, enabling the value to be reused.

Syntax

<i3ce_declare i3ce_name="*variable_name***" i3ce_value="***value***"**
i3ce_scope="*level***"/>**

Description

The ICE Server extends the availability of variables over standard HTML variables. HTML variables must be passed as part of the invoking URI. A reference to the HTML variable is replaced by the actual text when the Web Deployment Option parses the file. Web Deployment Option variables are more convenient in that they are maintained by the server and are not part of the Web Deployment Option address. Also, Web Deployment Option variables cannot have their values altered by a user changing the value in the Web Deployment Option address (URI) in the browser window.

The following table lists the syntax elements used with the <i3ce_declare> macro tag:

Syntax Element	Description
i3ce_name	Specifies the name of a variable.
i3ce_value	Specifies the value assigned to the variable.
i3ce_scope	Specifies the lifetime of the variable. The available choices are: server—loaded when the ICE Server starts and available for use until the ICE Server is shut down. A server variable persists as long as the repository database is not deleted. session—available for use while the Web user is logged in and has not timed out. page—available for use anywhere in the document. Once the document has been passed back to the browser, the variable becomes invalid.

Because all the variable lifetime values use the same name space, they are all accessed in the same way. For example:

```
:ServerVariable
:SessionVariable
:PageVariable
```

The declared variable can be used in a document by referring to it in a Web Deployment Option tag. Any variable can be used by preceding its name with the colon (:), but you cannot have a session-level variable with the same name as a server-level variable. This is why it is a good idea to establish a naming convention such as that suggested in the section Committing Transactions on the Home Page of the chapter "Creating Web Applications: An Example."

Example

This example declares a page level variable containing the text: Static Text"

```
<i3ce_declare i3ce_name="var1" i3ce_scope="page" i3ce_value="Static Text"/>
```

This example declare a session level variable containing the contents of the variable anotherVariable:

```
<i3ce_declare i3ce_name="var2" i3ce_scope="session"
i3ce_value=":anotherVariable"/>
```

See Also

<i3ce_var>

<i3ce_extend> Tag

Purpose

This tag enables the page designer to intermix HTML elements with Web Deployment Option elements to extend the functionality provided by the Web Deployment Option. The <i3ce_extend> tag is able to incorporate any content whenever it is used. The designer can use this to take closer control over the HTML that is generated by the Web Deployment Option. As a result it is up to the designer to ensure that the resultant HTML is syntactically correct.

Syntax

<i3ce_extend i3ce_tagName="*tag_to_be_inserted***">**
<i3ce_Attributes>
<i3ce_Attribute>
<i3ce_AttributeName>*tag_attribute_name***</i3ce_Attribute Name>**
<i3ce_AttributeValue>*tag_attribute_value***</i3ce_Attribute Value>**
</i3ce_Attribute>
</i3ce_Attributes>
<i3ce_Children>
<i3ce_Child>*child***</i3ce_Child>**
</i3ce_Children>
</i3ce_extend>

Description

Use the <i3ce_extend> element to build custom designed HTML constructs or even constructs that would be illegal at design time but legal once Web Deployment Option has processed the page. You must take great care using this element. The tag that is to be inserted is named in the i3ce_tagName attribute. This tag can have attributes and children. The children can be other tags or text. Attributes are further specified by setting both name and value using the <i3ce_AttributeName> and <i3ce_AttributeValue> tags, respectively. Any children or text can be individually inserted in the children list one per child element in the <i3ce_children> and <i3ce_child> tags respectively.

Example

The following example introduces an HTML anchor tag, which links to the Ingres home page on the Internet. The link is customized by giving it a red background.

```
<i3ce_extend i3ce_tagName="A">
    <i3ce_Attributes>
        <i3ce_Attribute>
            <i3ce_AttributeName>HREF</i3ce_AttributeName>
            <i3ce_AttributeValue>http://www.ingres.com</i3ce_AttributeValue>
        </i3ce_Attribute>
        <i3ce_Attribute>
            <i3ce_AttributeName>bgcolor</i3ce_AttributeName>
            <i3ce_AttributeValue>"red"</i3ce_AttributeValue>
        </i3ce_Attribute>
    </i3ce_Attributes>
    <i3ce_Children>
        <i3ce_Child>Link to Ingres home page</i3ce_Child>
    </i3ce_Children>
</i3ce_extend>
```

See Also

<i3ce_query>

<i3ce_function> Tag

Purpose

Invokes the specified Web Deployment Option user extension function.

Syntax

<i3ce_function i3ce_name="*function_name***"**
i3ce_location="*library_name***"**
[**<i3ce_parameters>**
<i3ce_parameter i3ce_name="*parameter_name***"**
i3ce_value="*parameter_value***"/>**...
</i3ce_parameters>]
[**<i3ce_HTML>***markup_text_with_variables* **</i3ce_HTML>**]
</i3ce_function>

Description

The following table lists the syntax elements used with the <i3ce_function> tag:

Syntax Element	Description
i3ce_name	Specifies the name of the function.
i3ce_location	Specifies the name of the library containing the function. It can be either a DLL (on Windows) or shared library (on UNIX).
i3ce_name	Specifies the name of a parameter being passed to the function.
i3ce_value	Specifies the value assigned to the parameter.
<i3ce_HTML>	Specifies any allowable markup text. A format string containing markup tags and column names. This option provides the ability to describe a line of markup language syntax and embed within it variable placeholders.

Example

```
<i3ce_function i3ce_name="func1"
    i3ce_location="lib1">
    <i3ce_parameters>
        <i3ce_parameter i3ce_name="p1" i3ce_value="v1"/>
        <i3ce_parameter i3ce_name="p2" i3ce_value="v2"/>
    </i3ce_parameters>
    <i3ce_HTML>
        <p>Random HTML & text including Web Deployment Option variables. e.g.:
:p2, :p1</p>
    </i3ce_HTML>
</i3ce_function>
```

See Also

<i3ce_query>, <i3ce_function>

<i3ce_if> Tag

Purpose

Evaluates a conditional expression.

Syntax

<i3ce_if> <i3ce_condition i3ce_condop="*conditional_operator***"**
i3ce_condlhs="*left_hand_side***" i3ce_condrhs="***right_hand_side***"/>**
<i3ce_then>
…
Text for then-branch
…
</i3ce_then>
[**<i3ce_else>**
…
Text for else-branch
…
<i3ce_else>]
<i3ce_if>

Description

The <i3ce_if tag> can test the value of a conditional expression. It has three child tags, <i3ce_condition>, <i3ce_then>, and <i3ce_else>. The first two tags are mandatory, and the last one is optional. The <i3ce_condition> tag has three attributes to specify a conditional expression. Depending on the results of evaluating this expression, either the <i3ce_then> or the <i3ce_else> branch is activated. The <i3ce_condition> tag attributes are described in the table below.

The following table lists the syntax elements used with the <i3ce_if macro> tag:

Syntax Element	Description
i3ce_condop	Specifies a conditional operator, one of "==", "!=", " <", or ">"
i3ce_condlhs	Specifies the left-hand side of the conditional expression.
i3ce_condrhs	Specifies the right-hand side of the conditional expression.
<i3ce_then>	Specifies the text that is activated if the condition evaluates to "true".
<i3ce_else>	Specifies the text that is activated if the condition evaluates to "false".

Comparisons are performed as string compares.

Comparison Operator	Description
= =	Equal
!=	Not equal
<	Less than
>	Greater than
>	Greater than

Example

```
<i3ce_if> <i3ce_condition i3ce_condop = "==" i3ce_condlhs=":VariableA"
    i3ce_condrhs="String"/>
        <i3ce_then>The expression evaluates to true, the variable 'VariableA'
            contains the text 'String'.</i3ce_then>
        <i3ce_else>The expression evaluates to false.</i3ce_else>
</i3ce_if>
```

```
<i3ce_if> <i3ce_condition i3ce_condop="=="i3ce_condlhs=":var1"
i3ce_condrhs=":p1"/>
    <i3ce_then> then text
    </i3ce_then>
    <i3ce_else> else text
    </i3ce_else>
</i3ce_if>
```

See Also

<i3ce_switch>

<i3ce_include> Tag

Purpose

Includes a specified fragment/page or macro fragment/page into the current document.

Syntax

<i3ce_include i3ce_name="*document_name***"**
i3ce_location="*business_unit_name***"**
i3ce_process="true"|"false">
<i3ce_parameters>
<i3ce_parameter i3ce_name="*parameter_name***"**
i3ce_value="*parameter_value***"/>**
</i3ce_parameters>
</i3ce_include>

Description

The <i3ce_include> tag enables you to include other Web Deployment Option macro documents into the current document. This promotes code reuse: you could, for example, design a common menu to be included on every page. You could collect variable definitions into a single page and include that at the start of a user's session. Furthermore, you can pass parameters on to the document that is being included; this allows you to pass parameters to the included document thus altering its behavior.

The following table lists the syntax elements used with the <i3ce_include> tag:

Syntax Element	Description
i3ce_name	Specifies the name of the document to be included.
i3ce_location	Specifies the name of the business unit containing the document to be included.
i3ce_process	If "true" is specified, the include file will also be processed by Web Deployment Option, interpreting any macros as required. If "false" is specified, the include file will not be processed by Web Deployment Option.
i3ce_name	Specifies the name of a parameter being passed to the included document.
i3ce_value	Specifies the value assigned to the parameter.

Example

This example shows the cascading style sheet file that is included in most of the documents within the plays business unit:

```
<i3ce_include i3ce_name="play_styleSheet.css" i3ce_location="plays">
    <i3ce_parameters></i3ce_parameters> </i3ce_include>
```

The following code sample is included in most of the Globe Shop documents. It displays an action bar that has various parts activated under parameter (variable) control:

```
<i3ce_include i3ce_name="play_shopAction-h.HTML" i3ce_location="plays">
    <i3ce_parameters><i3ce_parameter i3ce_name="View" i3ce_value="Yes"/>
    </i3ce_parameters>
</i3ce_include>
```

SeeAlso

i3ce_function

<i3ce_query> Tag

Purpose

Executes the SQL query provided and returns the result as specified.

Syntax

```
<i3ce_query
    [i3ce_database="database_name"]
    [i3ce_vnode="vnode_name"]
    [i3ce_class="connection_class"]
    [i3ce_user="user_name"]
    [i3ce_password="password"]>
    <i3ce_sql
        [i3ce_transaction="transaction_name"]
        [i3ce_cursor="cursor_name"]>
        <i3ce_statement>SQL_statement</i3ce_statement>
        [<i3ce_rowsPerRequest i3ce_rowcount="row_number"/>]
        [<i3ce_links>
            <i3ce_link i3ce_column="link_column_name"
                i3ce_target="link_target_URI"/>
                ...
        </i3ce_links>]
        [<i3ce_headers>
            <i3ce_header i3ce_column="relation_column_name"
            i3ce_text="replacement_name_for_column"/>
            ...
        </i3ce_headers> ]
        [<i3ce_attribute>HTML_attribute_specification
</i3ce_attribute>]
        [<i3ce_extension i3ce_name="file_extension"/>]
        [<i3ce_nullvar i3ce_value="value_for_NULLS_on_insert"/>]
        [<i3ce_relation_display i3ce_typename=
            "i3ce_table"| "i3ce_selector" | "i3ce_plain" |
            "i3ce_unformatted" | "i3ce_xml" | "i3ce_xmlpdata"/> ]
    </i3ce_sql>
</i3ce_query>
```

Description

The i3ce_query element allows the user to specify an SQL statement. The parent i3ce_query element contains an i3ce_statement tag, which contains the query text. The <i3ce_query> statement contains a number of attributes that are used to specify options other than the current defaults.

Web Deployment Option executes the SQL statement and formats the results as specified by the i3ce_typename attribute of the <i3ce_relation_display> tag. If the statement is not a select, a message is displayed. The message can be specified using the ii_success_message and ii_error_message HTML variables. The SQL statement can contain parameter markers of the form :*variable*, where variable is a defined HTML variable. HTML variables are defined using the <INPUT> tag. Note that variables set in an HTML form are not defined until that form is submitted; variables defined in a form on the same page as a Web Deployment Option macro will not be defined.

The following table lists the syntax element used with the <i3ce_query> macro tag:

Syntax Element	Description
i3ce_database	Specifies the database to which the query will be directed.
i3ce_vnode	Specifies the vnode to which the query will be directed. This element is used in combination with the i3ce_database attribute.
i3ce_class	Specifies the name of one of the Ingres servers or Enterprise Access products. See the Standard Command Line Flags and Parameters section in the *Command Reference Guide*.
i3ce_user	Specifies the name of the Web Deployment Option database user with which to associate the query. Web Deployment Option maps this user name to an actual Ingres user to run the query. See Database Users in the chapter "Managing the Web Deployment Option." This option must be specified with the i3ce_password attribute as a pair. **Note:** This option is provided for backward compatibility with Ingres/ICE 2.0 and is depreciated.
i3ce_password	Specifies the password for the user specified with the i3ce_user attribute. **Note:** This option is provided for backward compatibility with Ingres/ICE 2.0 and is depreciated.
i3ce_transaction	Specifies a unique name for a transaction. See the i3ce_transaction option description in this section for more information.
i3ce_cursor	Specifies a unique name for a cursor. If not used, a cursor is created by specifying the number of rows required (using the **<**i3ce_typename> tag). This option can only be specified when associated with a transaction (that is, the i3ce_transaction attribute is also specified). There is a limitation of one cursor per transaction. See the i3ce_cursor option description in this section for more information.

Syntax Element	Description
<i3ce_statement**>**	Specifies one or more SQL statements.
<i3ce_rowsPerRequest>	Specifies the number of rows for retrieval with the cursor.
<i3ce_links>	Generates a hypertext link to the URI for each item in the column. This option can only be specified when the i3ce_typename attribute is *not* set to "i3ce_unformatted". See the i3ce_link option description in this section for more information.
<i3ce_headers**>**	Allows the definition of the text used in column headers. By default, the relational table column name is used. This option can only be specified when the i3ce_typename attribute is *not* set to "i3ce_unformatted".
<i3ce_attribute>	Specifies a string representing any valid HTML attribute in the context of the i3ce_typename option. This option can only be specified when the i3ce_typename attribute is *not* set to "i3ce_unformatted". See the <i3ce_attribute**>** option description in this section for more information.
<i3ce_extension>	Specifies an extension that overrides the extension used for the temporary file when referring to a binary object. It is only valid when the output of a query contains a single column of binary objects. This option applies to all extracted binary columns.
<i3ce_nullvar>	Specifies the text that should be used when retrieving data from a table and the column contains NULL values.
<i3ce_relation_display>	Specifies the type of HTML formatting for the output. See the i3ce_typename option description in this section for information on the valid choices.
<i3ce_HTML>	Specifies a format string containing markup tags and column names. When using this option, the i3ce_typename attribute *must* be set to "i3ce_unformatted". See the <i3ce_HTML> option description in this section for more information.

i3ce_typename Option

The i3ce_typename option specifies the type of HTML formatting for the output. The valid values are:

- i3ce_table (default)—formats the result rows as an HTML table. The column headers are the names of the result columns. Each table cell contains a single item in the result set. If the result set contains Binary Large Objects (BLOBs), Web Deployment Option writes the BLOBs to temporary files and generates <IMG> tags to refer to them, indicating that the files contain image data. This output type supports the <i3ce_links> option.
- i3ce_selector—formats the results using the HTML SELECT tag. If the query contains multiple columns, the columns in each row are concatenated. This output type does not support the <i3ce_links>option.
- i3ce_plain—formats each row of the result set as a paragraph. If the result set contains BLOBs, Web Deployment Option writes the BLOBs to temporary files and generates <IMG> tags to refer to them, indicating that the files contain image data. This output type is particularly useful for placing images on a page. This output type supports the <i3ce_links> option.
- i3ce_unformatted—outputs the data with no HTML formatting or separators. If the result set contains BLOBS, Web Deployment Option writes the BLOBs to temporary files and places the URIs of the files on the output page. This output type is useful when you want to embed references to BLOBs in another HTML tag, for example, to fetch a background image for a page from a database. This output type does not support the <i3ce_links> option.
- i3ce_xml—XML generated from the query is formatted according to the Ingres DTD and XML literal characters are converted into CDATA.
- i3ce_xmlpdata—XML generated from the query is formatted according to the Ingres DTD. The data is not processed and it is the responsibility of the developer to ensure the generated output is well formed and valid.

i3ce_transaction Attribute

The i3ce_transaction attribute allows the association of a name with a transaction.

When writing Web Deployment Option queries, "auto commit" is the default action. Queries are committed if they complete successfully. With applications that require browsing and selecting items from a list (like a shopping cart), it is necessary to maintain a transaction over many pages and only commit the transaction when the user has finished. A transaction is terminated with either the <i3ce_commit> or the <i3ce_rollback> option with the transaction name.

i3ce_cursor Attribute

The i3ce_cursor attribute specifies a unique name for a cursor. When used, a *named* cursor is created which allows the full result set to be displayed page-by-page until the transaction is ended. If not specified, an *anonymous* cursor is created and which is closed when the rows have been returned.

Using this attribute, the number of rows on a page is defined by the Web author. This reduces the volume of data that is transmitted and the amount of time the browser spends waiting for the data.

<i3ce_links> Element

The i3ce_links element contains a list of i3ce_link elements. Each i3ce_link element has two attributes, i3ce_column and i3ce_target. Each column (specified by i3ce_column) can by linked with a URL target (specified by i3ce_target) and the contents of the column are passed as a variable to the target. The variable name is an HTML variable name and the value is the value of the named column for the current (or chosen) row (from the DTD file).

<i3ce_attribute> Element

The <i3ce_attribute> element allows the user to change the appearance of the page by specifying HTML attributes that will be applied to the generated HTML. The Web Deployment Option does not parse the value—it simply passes it through to the output page.

Valid children for the i3ce_attribute element include any HTML that is legal in the context of the specified output type, specified by the i3ce_typename attribute:

Value of i3ce_typename Attribute	Use of <i3ce_attribute>Value
i3ce_table	Specify the table border width, color, cell spacing, alignment, or any of the other HTML table attributes.
i3ce_selector	Specify the name of the HTML variable into which the browser will place the selected value.
i3ce_plain	Specify the attributes for image output.
i3ce_unformatted	Not available.

<i3ce_HTML> Option

The <i3ce_HTML> option allows the developer to include a line of HTML or markup language with embedded variable placeholders. This enables a developer to program using HTML tools that provide WYSIWYG rendering. The Web Deployment Option macros can then be added using the HTML already generated.

This removes dependence of Web Deployment Option on knowledge of HTML or other markup syntax when building output.

Example

The following example selects all columns from the table plays in the iceTutor database. It creates a transaction and a cursor. It retrieves 5 rows at a time and formats them into an HTML table.

```
<i3ce_query i3ce_database="iceTutor">
    <i3ce_sql i3ce_transaction="myTransaction" i3ce_cursor="myCursor">
        <i3ce_statement>
            select * from plays
        </i3ce_statement>
        <i3ce_rowsPerRequest i3ce_rowcount="5"/>
        <i3ce_relation_display i3ce_typename="i3ce_table"/>
    </i3ce_sql>
</i3ce_query>
```

This example code selects the play, performed date, and author name from the plays table. It creates links based on the author and play names to the template files Authors.HTML and play_types.HTML, respectively. It also changes the header of the 'performed' column to be "Date of First Performance". The attribute tag sets the table border width to be "3".

```
<i3ce_query i3ce_database="playsdb" i3ce_vnode="globe"
    i3ce_user="user_name" i3ce_password="password">
    <i3ce_sql i3ce_transaction="Complete" i3ce_cursor="Works">
    <i3ce_statement>
        select name as 'Play Name', performed, playwright as 'Author', type
        from plays
    </i3ce_statement>
    <i3ce_rowsPerRequest i3ce_rowcount="5"/>
    <i3ce_links>
        <i3ce_link i3ce_column="Author" i3ce_target=
        "http://www.globe.com/ice-bin/oiice.dll/Author_gp[Authors.HTML]"/>
        <i3ce_link i3ce_column="type" i3ce_target="plays[play_types.HTML]"/>
    </i3ce_links>
    <i3ce_headers>
        <i3ce_header i3ce_column="performed" i3ce_text="Date of First
            Performance"/>
    </i3ce_headers>
    <i3ce_attribute>border=3</i3ce_attribute>
    <i3ce_relation_display/>
    </i3ce_sql>
</i3ce_query>
```

See Also

<i3ce_commit>, <i3ce_rollback>, <i3ce_function>, <i3ce_include>

<i3ce_rollback> Tag

Purpose

Rolls back a previously started transaction.

Syntax

<i3ce_rollback i3ce_transaction="*transaction_name*"/>

Description

The <i3ce_rollback> tag rolls back the transaction specified by *transaction_name*. The transaction name must have been defined previously using the i3ce_transaction option of the <i3ce_sql> tag.

Note: This is the default action for a session that times out.

Example

```
<i3ce_rollback i3ce_transaction="myTransaction"/>
```

See Also

<i3ce_query>, <i3ce_commit>

<i3ce_switch> Tag

Purpose

Tests the value of an expression against a number of constant values and executes an associated expression based on the value that matches.

Syntax

<i3ce_switch i3ce_value="*switch_expression***">**
    <i3ce_case i3ce_value="*constant*$_1$**">***Action for case 1***</i3ce_case>**
    ...
    <i3ce_case i3ce_value="*constant*$_n$**">***Action for case n***</i3ce_case>**
    <i3ce_default>*Action for default case***</i3ce_default>**
</i3ce_switch>

Description

The following table lists the syntax elements used with the <i3ce_switch> macro tag:

Syntax Element	Description
switch_expression	The value you want to compare to the constant values. It is usually an expression containing one or more variables.
constant$_n$	A constant value to be compared to *switch_expression*.
action for case n	An action for the particular case. It can be any markup text, including variables or Web Deployment Option XML language commands.
action for default case	The default action, if none of the compared values match *switch_expression*.

Example

```
<i3ce_switch i3ce_value=":shape">
    <i3ce_case i3ce_value="T">Triangle </i3ce_case>
    <i3ce_case i3ce_value="S">Square </i3ce_case>
    <i3ce_case i3ce_value="P">Pentagon </i3ce_case>
    <i3ce_default>Circle </i3ce_default>
</i3ce_switch>
```

See Also

<i3ce_if>

<i3ce_var> Tag

Purpose

Replaces a variable within a string with its actual value.

Syntax

<i3ce_var i3ce_name="*variable_name***"/>**

Description

This tag can be used to replace the named variable, *variable_name*, with its textual value.

Example

```
<i3ce_var i3ce_name="myVariable"/>
```

See Also

<i3ce_declare>

Macro Statements

The Web Deployment Option allows macro statements to be embedded in HTML documents, specifying SQL statements that are executed and whose result sets are automatically formatted by Web Deployment Option.

Macro Statement Format

The format of a Web Deployment Option macro statement is as follows:

<!-- #ICE [*keyword*=`*value*`] **-->**

In this syntax representation:

- "**<!--**" and "**-->**" are the HTML comment delimiters
- **#ICE** is the Web Deployment Option macro marker
- *keyword* is a valid Web Deployment Option keyword
- *value* is the value assigned to the keyword, delimited by grave (back) quotes (`)

Web Deployment Option macros can be embedded anywhere in an HTML document. Because they are always contained within the HTML comment delimiters, Web Deployment Option macros remain valid in all HTML documents.

By using a grave quote as the delimiter, you are free to include both single (') and double quote (") characters in the values of macro language keywords. If you need to include a grave quote character in a macro value, double the grave quote character (``).

Note: All keywords except for SQL and VAR require a session. However, there are certain *options* for the SQL keyword that do require a session. For information about these SQL options, see the SQL Keyword in this chapter.

Macro Keywords

The following macro language keywords are available and are described in more detail in the sections that follow:

- COMMIT
- DECLARE
- FUNCTION
- IF
- INCLUDE
- ROLLBACK
- SQL
- SWITCH
- VAR

Macro Keywords

This section presents the syntax for each Web Deployment Option macro keyword.

COMMIT Keyword

Purpose

Commits a previously started transaction.

Syntax

<!-- #ICE COMMIT=`***transaction_name***`-->

Description

The COMMIT keyword commits the transaction specified by *transaction_name*.

The transaction name must have been defined previously using the TRANSACTION option of the SQL keyword.

Example

```
<!-- #ICE COMMIT=`myTransaction` -->
```

See Also

SQL keyword, ROLLBACK keyword

DECLARE Keyword

Purpose

Assigns a value to a named variable, enabling the value to be re-used.

Syntax

<!-- #ICE [**REPEAT**] **DECLARE=**`[*level*.]*variable_name*=*value*` **-->**

Description

The ICE Server extends the availability of variables over standard HTML variables. HTML variables must be passed as part of the invoking URI. A reference to the HTML variable is replaced by the actual text when Web Deployment Option parses the file.

The Web Deployment Option variables are more convenient in that they are maintained by the server and are not part of the Web Deployment Option address (URI). Also, Web Deployment Option variables cannot have their values altered by a user changing the value in the Web Deployment Option URI within the browser address window.

The following table lists the parameters used with the DECLARE keyword:

Parameter	Description
level	The lifetime of the variable. The valid values are: server—loaded when the ICE Server starts and available for use until the ICE Server is shut down. A server variable persists as long as the repository database is not deleted. session—available for use while the Web user is logged in and has not timed out. page—available for use anywhere within the document. Once the document has been passed back to the browser, the variable becomes invalid.
variable_name	The name of a variable.
value	The value assigned to the variable.

Because all the variable lifetime values use the same name space, they are all accessed in the same way. For example:

```
:ServerVariable
:SessionVariable
:PageVariable
```

The declared variable can be used within a document by referring to it in a Web Deployment Option macro. Any variable can be used by preceding its name with the colon (:), but you cannot have a session-level variable with the same name as a server-level variable. This is why it is a good idea to establish a naming convention such as that suggested in the section Committing Transactions on the Home Page of the chapter "Creating Web Applications: An Example."

REPEAT Option

The REPEAT option allows the recursive parsing of macros. This allows multiple queries and macros to be embedded within a single macro. It must be used in conjunction with another macro option and has no meaning when used on its own.

Since the grave quote (`) and the colon (:) characters are used as delimiters, they must be duplicated when used with the REPEAT option. This option has the effect of disabling the delimiting effect when used with the REPEAT option to preserve the delimiting effect.

Note: The REPEAT option must be used carefully with the DECLARE keyword. It implies that further macros must be resolved, the results of which will be used by the DECLARE keyword. It is possible to declare variables containing large strings or to issue a repeated declare that recurses and takes memory. All declared information will permanently reside in memory.

Examples

```
<!-- #ICE DECLARE=`page.myVar=static string` -->
<!-- #ICE DECLARE=`page.myVar=:anotherVariable` -->
<!-- #ICE DECLARE=`page.myVar=:anotherVariable +
    static string` -->
<!-- #ICE DECLARE=`session.myVar=static string` -->
<!-- #ICE DECLARE=`session.myVar=:anotherVariable`-->
<!-- #ICE REPEAT DECLARE=`session.myRepeatVar==`<!-- #ICE SQL=``select BgColor
    from Style where Style_id = 4`` -->`
```

FUNCTION Keyword

Purpose

Invokes the specified Web Deployment Option extension function or server function.

Syntax

For Extension Functions

<!-- #ICE [REPEAT] FUNCTION=
`[*library_name*.]*function_name*?{*variable_name*=*value*}[&...]`
[**HTML=**`*HTML text with variables*`]
-->

For Server Functions

<!-- #ICE [REPEAT]
FUNCTION=`*server_function_name*?**action=***action*
[&*property*=*value*][{&*property*=*value*}]`
-->

Description

Parameters

The following table lists the parameters used with the FUNCTION keyword:

Parameter	Description
library_name	The name of the library containing the function. It can be either a DLL (on Windows NT) or shared library (on UNIX). **Note:** This parameter should not be included when specifying server functions.
function_name	The name of the function.
variable_name	The name of a variable being passed to the function.

Parameter	Description
value	The value assigned to the variable. For server functions, *value* is only specified if the requested action/property combination is associated with an output value.
action	For ICE Server functions, specifies the query operation to be executed. Possible values include select, retrieve, insert, update, and delete, depending on the function. (For valid action parameters, see "Appendix D: ICE Server Functions.")
property	The ICE Server parameter to which the action applies. (For valid property values, see "Appendix D: ICE Server Functions.")
HTML text with variables	Any allowable HTML text. A format string containing markup tags and column names. This option provides the ability to describe a line of HTML or markup language syntax and embed within it variable placeholders.

Also, for a description of the possible functions and variables that can be used with the FUNCTION keyword to access the ICE Server, see "Appendix D: ICE Server Functions." Server functions can also be invoked through the Web Deployment Option C API; for more information, see "Chapter 7: Using the C API."

REPEAT Option

The REPEAT option allows the recursive parsing of macros. This allows multiple queries and macros to be embedded within a single macro. It must be used in conjunction with another macro option and has no meaning when used on its own.

Since the grave quote (`) and the colon (:) characters are used as delimiters, they must be duplicated when used with the REPEAT option. This option has the effect of disabling the delimiting effect when used with the REPEAT option to preserve the delimiting effect.

Example

The following example invokes the "unit" ICE Server function, which changes the play's unit ID to 3 and unit name to Shakespeare:

```
<!-- #ICE FUNCTION=`unit?action=
    update&unit_id=3&unit_name=Shakespeare`
-->
```

IF Keyword

Purpose

Evaluates a conditional expression.

Syntax

```
<!-- #ICE [REPEAT]
        IF `condition`
        THEN=`...`
        [ELSE=`...`]
-->
```

Description

Parameters

The following table lists the parameters used with the IF keyword:

Parameter	Description
condition	A single or a compound conditional expression where: *condition* = *comparison* {**AND** | **OR**} *comparison* and: *comparison* = `...` { **==** | **!=** | **<** | **>** } `...` or **DEFINED(***variable_name***)**

Conditional expressions are available to allow HTML output that is dependent on the result of an expression. An expression is composed of one or more comparisons. Comparisons are performed as string compares. The comparison operators are:

Comparison Operator	Description
==	Equal
!=	Not equal
<	Less than
>	Greater than

Multiple comparisons are expressed using logical operators:

Logical Operator	Description
AND	Logical AND
OR	Logical OR

Existence Function	Description
DEFINED(*variable_name***)**	Returns true if *variable_name* is defined.

REPEAT Option

For information on the REPEAT option, see the FUNCTION Keyword in this chapter.

Examples

```
<!-- #ICE IF ( `:VariableA` == `String` )
    THEN=`<b>The expression evaluates to true.</b>`
    ELSE=`<b>The expression evaluates to false.</b>`
-->

<!-- #ICE IF (DEFINED(VariableA))
    THEN=`<b>The variable exists</b>`
    ELSE=`<b>The variable doesn't exist.</b>`
-->

<!-- #ICE REPEAT IF (`:ii_status_number` == `0`)
    THEN=`<!-- #ICE INCLUDE=``success.HTML`` -->`
-->
```

See Also

SWITCH keyword

INCLUDE Keyword

Purpose

Includes a Web Deployment Option HTML or macro document into the current document.

Syntax

<!-- #ICE [**REPEAT**] **INCLUDE=**`[*business_unit_name*
'[']*document_name*[']']**?**{*variable_name***=***value*}[**&**...]`
[**TYPE=**`**HTML**|**MULTI**|**REPORT**|**EXE**`]
-->

Description

Parameters

The following table lists the parameters used with the INCLUDE keyword:

Parameter	Description
business_unit_name	The name of the business unit containing the document to be included.
document_name	The name of the document to be included.
variable_name	The name of the variable.
value	The value assigned to the variable.

TYPE Option

The TYPE option is used to distinguish the action the ICE Server should take when processing the included file. This allows reuse of common component documents.

There are four selections when using the TYPE option with the INCLUDE keyword:

- HTML (pages)

  This is the default option. When you include a page, the document can access every page variable defined in the current page. You can add new parameters in the include call. If you include a page with no REPEAT, the user must be granted read permission for this document. If you include a page with a REPEAT, the user must be granted execute and read permissions for this document.

- MULTI (facets)

  To include a facet, the user must be granted the execute permission for the document.

- REPORT (reports)
- EXE (applications)

REPEAT Option

For information on the REPEAT option, see the FUNCTION Keyword in this chapter.

Examples

This example shows the cascading style sheet file that is included in most of the documents within the plays business unit:

```
<!-- #ICE INCLUDE=`plays[play_styleSheet.css]`
    TYPE=`MULTI` -->
```

The following code sample is included in most of the Globe Shop documents. It displays an action bar that has various parts activated under parameter (variable) control:

```
<!-- #ICE REPEAT INCLUDE=
    `plays[play_shopAction_h.HTML]?View=Yes`-->
```

ROLLBACK Keyword

Purpose

Rolls back a previously started transaction.

Syntax

<!-- #ICE ROLLBACK=`***transaction_name***`-->

Description

The ROLLBACK keyword rolls back the transaction specified by *transaction_name*.

The transaction name must have been defined previously using the TRANSACTION option of the SQL keyword.

This is the default action for a session that times out.

Example

```
<!-- #ICE ROLLBACK=`myTransaction` -->
```

See Also

COMMIT keyword, SQL keyword

SQL Keyword

Purpose

Executes the SQL query provided and returns the result as specified.

Syntax

<!-- #ICE [**REPEAT**]
SQL=`*query*`
[**TYPE=`TABLE`**|`**SELECTOR`**|`**PLAIN`**|`**UNFORMATTED`**|
`XML`|`**XMLPDATA`]**
[**DATABASE=`***database_name*`]
[**TRANSACTION=`***transaction_name*`]
[**CURSOR=`***cursor_name*`]
[**ROWS=`***number_of_rows*`]
[**USER=`***user_name*`]
[**PASSWORD=`***password*`]
[**LINKS=`**{*column_name, URI*}[**,**…]`]
[**HEADERS=`**{*column_name, text*}[**,**…]`]
[**ATTR=`***attribute*`]
[**EXT=`***extension*`]
[**NULLVAR=`***text*`]
[**HTML=`***markup text with variables*`]
[**XML=`***markup text with variables*`]
[**XMLPDATA=`***markup text with variables*`]
-->

Description

The value of the SQL keyword, *query*, is specified by one or more SQL statements. The SQL keyword also provides a variety of options. The Web Deployment Option executes the SQL statements and formats the results as specified by the TYPE option. If the statement is not a select, a message is displayed. The message can be specified using the ii_success_message and ii_error_message HTML variables.

The SQL statement can contain parameter markers of the form **:***variable*, where *variable* is a defined HTML variable. HTML variables are defined using the <INPUT> tag. Note that variables set in an HTML form are not defined until that form is submitted; variables defined in a form on the same page as a Web Deployment Option macro will not be defined at the time the page is parsed by Web Deployment Option.

The TRANSACTION, CURSOR, and ROWS options to the SQL keyword require a session. A session is established when a user connects to and is authenticated by the ICE Server. The session lasts until either a timeout occurs or the user logs out. It is used by Web Deployment Option to maintain information about user context.

Syntax Elements

The following table lists the syntax elements used with the SQL keyword:

Syntax Element	Description
query	Specifies one or more SQL statements.
TYPE	Specifies the type of HTML formatting for the output. For information on the valid choices, see the TYPE option description in this section.
DATABASE	Specifies the database to which the query will be directed.
TRANSACTION	Specifies a unique name for a transaction. For more information, see the TRANSACTION option description in this section.
CURSOR	Specifies a unique name for a cursor. If not used, a cursor is created by specifying the number of rows required (using the ROWS option). This option can only be specified when associated with a transaction (that is, the TRANSACTION option is also specified). There is a limitation of one cursor per transaction. For more information, see the CURSOR option description in this section.
ROWS	Specifies the number of rows for retrieval with the cursor.
USER	Specifies the name of the Web Deployment Option database user with which to associate the query. The Web Deployment Option maps this user name to an actual Ingres user to run the query. See Database Users in the chapter "Managing the Web Deployment Option." This option must be specified with the PASSWORD option. **Note:** This option is provided for backward compatibility with Ingres/ICE 2.0 and is depreciated.

Syntax Element	Description
PASSWORD	Specifies the password for the user specified with the USER option. **Note:** This option is provided for backward compatibility with Ingres/ICE 2.0 and is depreciated.
LINKS	Generates a hypertext link to the URI for each item in the column. This option can only be specified when TYPE is not UNFORMATTED. For more information, see the LINKS option description in this section.
HEADERS	Allows the definition of the text used in column headers. By default, the relational table column name is used. This option can only be specified when TYPE is not UNFORMATTED.
ATTR	Specifies a string representing any valid HTML attribute in the context of the TYPE option. This option can only be specified when TYPE is not UNFORMATTED. For more information, see the ATTR option description in this section.
EXT	Specifies an extension that overrides the extension used for the temporary file when referring to a binary object. It is only valid when the output of a query contains a single column of binary objects. This option applies to all extracted binary columns.
NULLVAR	Specifies the text that should be used when retrieving data from a table and the column contains NULL values.
HTML	Specifies a format string containing markup tags and column names. When using this option, the TYPE option must be set to UNFORMATTED. For more information, see the HTML option description in this section.
XML	Specifies a format string containing markup tags and column names. The variable data is processed and XML literal characters are converted into CDATA.

Syntax Element	Description
XMLPDATA	Specifies a format string containing markup tags and column names. The variable data is not processed and any XML literal characters are left unchanged. It is the responsibility of the developer to ensure that the resulting generated XML is well formed and valid.

TYPE Option

The TYPE option specifies the type of HTML formatting for the output. The valid values are:

- TABLE (default)—formats the result rows as an HTML table. The column headers are the names of the result columns. Each table cell contains a single item in the result set. If the result set contains Binary Large Objects (BLOBs), Web Deployment Option writes the BLOBs to temporary files and generates <IMG> tags to refer to them, indicating that the files contain image data. This output type supports the LINKS option.
- SELECTOR—formats the results using the HTML SELECT tag. If the query contains multiple columns, the columns in each row are concatenated. This output type does not support the LINKS option.
- PLAIN—formats each row of the result set as a paragraph. If the result set contains BLOBs, Web Deployment Option writes the BLOBs to temporary files and generates <IMG> tags to refer to them, indicating that the files contain image data. This output type is particularly useful for placing images on a page. This output type supports the LINKS option.
- UNFORMATTED—outputs the data with no HTML formatting or separators. If the result set contains BLOBS, Web Deployment Option writes the BLOBs to temporary files and places the URIs of the files on the output page. This output type is useful when you want to embed references to BLOBs in another HTML tag, for example, to fetch a background image for a page from a database. This output type does not support the LINKS option.
- XML—the XML generated from the query is formatted according to the Ingres DTD and XML literal characters are converted into CDATA.
- XMLPDATA—the XML generated from the query is formatted according to the Ingres DTD. The data is not processed and it is the responsibility of the developer to ensure that the generated output is well formed and valid.

TRANSACTION Option

The TRANSACTION option allows the association of a name with a transaction.

When writing Web Deployment Option queries, "auto commit" is the default action. Queries are committed if they complete successfully. With applications that require browsing and selecting items from a list (like a shopping cart), it is necessary to maintain a transaction over many pages and only commit the transaction when the user has finished. A transaction is terminated with either the COMMIT or the ROLLBACK option with the transaction name.

CURSOR Option

The CURSOR option specifies a unique name for a cursor. When used, a *named* cursor is created which allows the full result set to be displayed page-by-page until the transaction is ended. If not specified, an *anonymous* cursor is created and which is closed when the rows have been returned.

Using this option, the number of rows on a page is defined by the Web author. This reduces the volume of data that is transmitted and the amount of time the browser spends waiting for the data.

LINKS Option

The value of the LINKS option has the form `` `*column_name*,*URI*` ``. For example:

```
`type,http://www.foo.com/typeinfo.HTML`
```

This would generate links to the URI, http://www.foo.com/typeinfo.HTML, for each item in the type column in the result set.

You can specify as many comma-separated column and URI pairs as you require. As Web Deployment Option processes the result set, for each item in a LINKS column, it generates a hyperlink tag to the specified URI for each item in the column.

To enable the referenced page to determine which item was clicked on, Web Deployment Option sets an HTML variable. The variable has the same name as the column name, and its value is the value clicked on. The referenced page can use this variable, typically by making it a parameter in another SQL statement. You cannot generate links for a BLOB column.

ATTR Option

The ATTR option allows the user to change the appearance of the page by specifying HTML attributes that will be applied to the generated HTML. The Web Deployment Option does not parse the value—it simply passes it through to the output page.

Valid values for the ATTR option include any HTML that is legal in the context of the specified output type, specified by the TYPE option:

Value of TYPE Option	Use of ATTR Option
TABLE	Specify the table border width, color, cell spacing, alignment, or any of the other HTML table attributes.
SELECTOR	Specify the name of the HTML variable into which the browser will place the selected value.
PLAIN	Specify the attributes for image output.
UNFORMATTED	Not available.

HTML Option

The HTML option allows the developer to include a line of HTML or markup language with embedded variable placeholders. This enables a developer to program using HTML tools that provide WYSIWYG rendering. The Web Deployment Option macros can then be added using the HTML already generated.

This removes dependence of Web Deployment Option on knowledge of HTML or other markup syntax when building output.

REPEAT Option

For information on the REPEAT option, see the FUNCTION Keyword in this chapter.

Examples

Example 1

The following examples show several different uses of the SQL macro keyword and a sample of the source output:

```
<!-- #ICE
    SQL=`select * from icetable`
    DATABASE = `iceTutorial`
    TRANSACTION=`myTransaction`
    CURSOR=`myCursor`
    ROWS=`10`
    TYPE=`TABLE`
-->

<!-- #ICE
    SQL=`select * from icetable`
    DATABASE = `iceTutorial`
    TRANSACTION=`myTransaction`
    CURSOR=`myCursor`
    ROWS=`10`
    TYPE=`TABLE`
    HEADERS=`i_title,Category`
    LINKS=`i_title,www.uri.com`
    ATTR=`border=1`
-->

<!-- #ICE
    SQL = `select title,lastname from
        book,author,bookauthor where book.bookid =
        bookauthor.bookid and bookauthor.authid =
        author.authid`
-->
```

An example of the generated output produced by the ICE Server follows:

```
<TABLE><TR>
<TH>title</TH>
<TH>lastname</TH>
</TR>
<TR>
<TD>Hamlet
<TD>Shakespeare
</TR>
<TR>
<TD>Macbeth
<TD>Shakespeare
</TR>
<TR>
</TABLE>
```

Example 2

This example shows how the LINKS option can be used in conjunction with the SQL macro keyword to produce a parameterized list of links:

```
<!-- #ICE
        DATABASE = `icetutor`
        SQL=`select distinct type from plays`
        TYPE=`PLAIN`
        LINKS=`type,/ice-bin/oiice.dll/my_playgroup/
        my_plays[myplay_typeLinkSubSet.HTML]`
-->
```

The HTML generated by the ICE Server follows:

```
<A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLinkSubSet.HTML]?type=comedy">
    comedy</A>

<A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLinkSubSet.HTML]?type=history">
    history</A>

<A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLinkSubSet.HTML]?type=tragedy">
    tragedy</A>
```

Example 3

The following example shows the results of the query formatted for XML according to the Ingres DTD.

```
<!-- #ice database=`icetutor`
    sql=`select * from plays`
    type=`xml`
-->
```

The generated output produced follows:

```
<?xml: version='1.0' ?>

<resultset>
<row>
<column column_name="comporder">1</column>
<column column_name="title">The Two Gentlemen of Verona</column>
<column column_name="playwright">Shakespeare</column>
<column column_name="performed">1598</column>
<column column_name="acts">5</column>
<column column_name="type">comedy</column>
</row>
```

```
•<column column_name="comporder">37</column>
<column column_name="title">Henry VIII</column>
<column column_name="playwright">Shakespeare</column>
<column column_name="performed">1613</column>
<column column_name="acts">5</column>
<column column_name="type">history</column>
</row></resultset>
```

Example 4

The following example shows the retrieval of XML data stored as a regular text field as child tags of an XML result set. It is the responsibility of the developer to ensure that the generated XML is well-formed.

```
<!-- #ice database=`icetutor`
    sql=`select * from test`
    type=`xmlpdata`
-->
```

The generated output produced follows:

```
<?xml: version='1.0' ?>

<resultset>
<row>
<column column_name="idx">1</column>
<column column_name="xmltest">
<plays>
<play>
<comporder>1</comporder>
<title>The Two Gentlemen of Verona</title>
<playwright>Shakespeare</playwright>
<performed>1598</performed>
<acts>5</acts>
<type>comedy</type>
</play>

<play>
<comporder>5</comporder>
<title>Titus Andronicus</title>
<playwright>Shakespeare</playwright>
<performed></performed>
<acts>5</acts>
<type>tragedy</type>
</play>
</plays>
</column>
</row>
</resultset>
```

Example 5

The following example shows the results of the query formatted for XML according to the defined markup:

```
<plays>
<!-- #ice database=`icetutor`
    sql=`select comporder, title, playwright, performed, acts, type from plays`
    xml=`<play>
<comporder>:comporder</comporder>
<title>:title</title>
<playwright>:playwright</playwright>
<performed>:performed</performed>
<acts>:acts</acts>
<type>:type</type>
</play>`
-->
</plays>
```

The generated output produced follows:

```
<?xml: version='1.0' ?>

<plays>
<play>
<comporder>1</comporder>
<title>The Two Gentlemen of Verona</title>
<playwright>Shakespeare</playwright>
<performed>1598</performed>
<acts>5</acts>
<type>comedy</type>
</play>

<play>
<comporder>37</comporder>
<title>Henry VIII</title>
<playwright>Shakespeare</playwright>
<performed>1613</performed>
<acts>5</acts>
<type>history</type>
</play>
</plays>
```

See Also

COMMIT keyword, ROLLBACK keyword

SWITCH Keyword

Purpose

Tests the value of an expression against a number of constant values and executes an associated expression based on the value that matches.

Syntax

<!-- #ICE [**REPEAT**]
SWITCH=`*switch_expression*`
CASE `*constant*$_1$`**=**`*result*$_1$`
CASE `*constant*$_2$`**=**`*result*$_2$`
…
CASE `*constant*$_n$`**=**`*result*$_n$`
[**DEFAULT=**`*default_result*`]
-->

Description

The following table lists the parameters used with the SWITCH keyword:

Parameter	Description
switch_expression	The value you want to compare to the constant values. It is usually an expression containing one or more variables.
constant$_n$	A constant value to be compared to the *switch_expression*.
result$_n$	A resulting value. It can be any markup text, including variables and Web Deployment Option macro commands.
default_result	The default resulting value, if none of the compared values match the *switch_expression*.

REPEAT Option

For information on the REPEAT option, see the FUNCTION Keyword in this chapter.

Example

The following example results in the string "Pentagon" appearing in the document if the variable product is set to "P":

```
<!-- #ICE SWITCH= `:shape`
    CASE `T`=`Triangle`
    CASE `S`=`Ingres`
    CASE `P`=`Pentagon`
    DEFAULT=`Circle`
-->
```

See Also

IF keyword

VAR Keyword

Purpose

Replaces a variable within a string with its actual value.

Syntax

<!-- #ICE [**REPEAT**] **VAR=**`*HTML text with variables*` **-->**

Description

HTML text with variables is any allowable HTML text. Variables are denoted by preceding the variable name with a colon (:).

This keyword can be used to read the specified variables and replace them with their actual values within a text string.

REPEAT Option

The REPEAT option causes a reparse of the resultant string after the variables have been inserted. This allows variables to contain macros or other variables. For information on the REPEAT option, see the FUNCTION Keyword in this chapter.

Example

In the Plays tutorial application, the HTML containing the VAR macro keyword appears as follows, using the variable e_orderNumber:

```
<!-- #ICE VAR=`Your order number
    <b>:e_orderNumber</b> will now be processed.<br>
    Please quote this number in all correspondence`
-->
```

The application sets the e_orderNumber session variable to a character string that becomes part of the message to the customer.

See Also

DECLARE keyword

Chapter 6: Creating Web Applications: An Example

In this chapter, you will learn how to create Web applications using Web Deployment Option and standard HTML programming concepts. You will learn how to recreate the Plays tutorial application.

The chapter takes an in-depth look at some of the various steps you may go through in creating a Web Deployment Option application. To begin, we create some basic Web Deployment Option objects and HTML files, and register them with the ICE server. From there, we explain the programming concepts used in constructing the pages for the Plays application.

Note: This chapter can be used in a variety of ways. It can be used as a tutorial, whereby you actually create the objects and files as you go—or you can simply follow along and learn about the features of Web Deployment Option. Alternatively, you may want to read certain sections only to apply specific concepts to your Web Deployment Option application.

The following topics are covered:

- Creating your application files and server location, and registering them with Web Deployment Option
- Setting up security for Web Deployment Option using a variety of objects including a session group, business unit, database connection, and others
- Designing the pages for your application, using the following Web Deployment Option programming features:
  - Automatic user account creation
  - SQL language support
  - Transaction support (commit/rollback)
  - Cursor support
  - Web Deployment Option native variables
  - Control flow statements
  - Automatic HTML code generation
  - Automatic hyperlink generation
  - Automatic HTML support for BLOBs (Binary Large OBjects)
  - Advanced security model
  - Fine-tuning of generated HTML
  - Parameterized include mechanism, promoting code reuse

- C language function support
- IMA (Ingres Management Architecture) support

Tip: You can access the online HTML-based Plays Tutorial application or the online Tutorial Guide by accessing the address http://*your_machine_name*/ice_index.html.

Before You Begin

This chapter assumes you have a basic understanding of HTML and its components. Standard HTML concepts are not reviewed as part of this tutorial—only Web Deployment Option features are examined in detail. If you would like to refresh your knowledge of HTML, see "Appendix B: HTML Primer."

If you do intend to work through the creation of an application, which is a duplicate of the Plays tutorial application, be sure you have your HTTP server and Web Deployment Option installed and running. You should be logged in as the Web Deployment Option privileged user so that you can perform all of the functions in this chapter. Also, take a moment to familiarize yourself with the pages in the ice subdirectory of your Ingres installation. You will be recreating many of these pages.

Finally, ensure that you know how to address your ICE server (for example, http://*your_machine_name*/ice-bin/oiice.dll).

A Tour of the Plays Application

The Plays application allows a visiting Web user to browse through the works of William Shakespeare at the Globe Centre for Shakespeare studies. After viewing Shakespeare's works in a variety of ways and based on different criteria, visitors can go shopping for some of their favorite souvenirs at the online Globe Boutique.

The vision for the Web site, as directed by the Art Director of the Globe Centre, is to provide a way for the Web user to select a group of Shakespeare's plays by type—that is, by choosing comedy, history, or tragedy. Additionally, the Web site should be visual so that icons should be used to select the play type. The Plays application, however, presents the different iterations of the development process, showing how the Web author accomplished this goal by the simplest means working up to the more sophisticated features of Web Deployment Option—the most elegant of which is the desired result.

Note: The Plays application is designed to demonstrate various implementations of the features of Web Deployment Option and is not necessarily intended to represent a "real-world" application. It represents the progression of the development of the site.

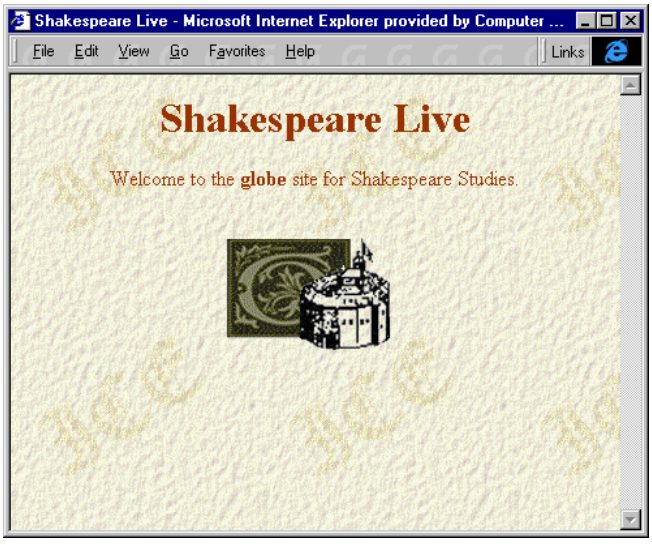
Let us now take a brief tour of the Plays application.

Plays Welcome Page

To begin, we enter the following address in the address bar, assuming that the name of the machine on which you are running Web Deployment Option is "globe":

```
http://globe/ice_index.html
```

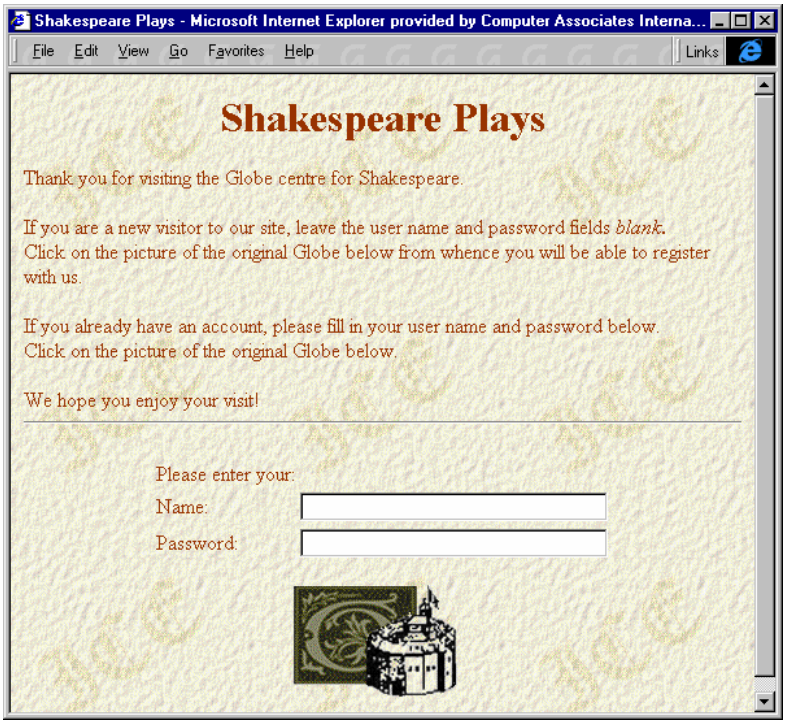
You will then see a list of options for Web Deployment Option. Select the Example Tutorial Application option. This invokes the welcome page of the Plays application:



Clicking the icon, we proceed to the login page for the application.

Plays Login Page

The Plays login page contains some introductory text that explains the login process. In addition, two controls are provided that enable the Web user to be authenticated to the ICE server, which provides access to the remaining pages in the Plays application.

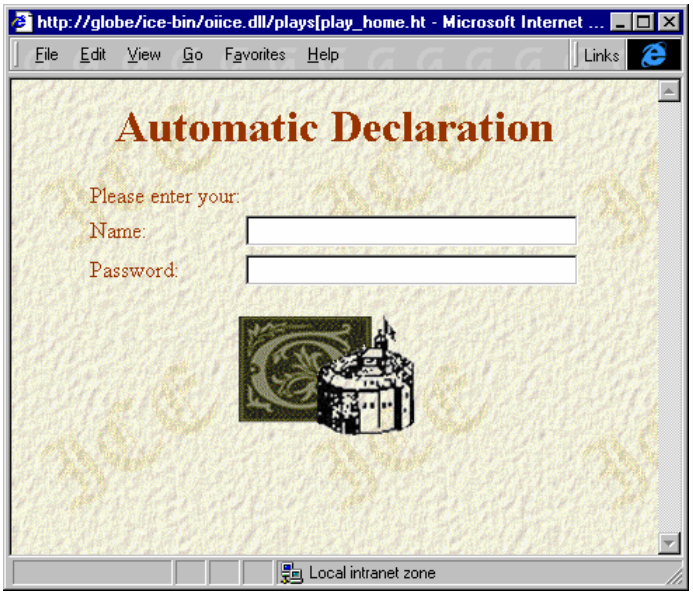


This page allows you to enter a name and password that you have already defined, or a new name and password. If the user is already defined, and you click the icon, you will proceed to the Home page of the application.

If the user name is unknown to the system, you are brought to an Automatic Declaration page, which allows you to define yourself as a user on the system. We will look at this page next.

Automatic Declaration Page

The Automatic Declaration page is the page on which you declare yourself as a Web user to the Web Deployment Option system:



Enter a unique user name in the Name edit control (do not use "ingres" or leave this blank). Then enter a password to be associated with your user name in the Password edit control. Next, click the icon to establish a connection to Web Deployment Option and proceed to the Plays home page.

Note that when you subsequently log in, you will already be authenticated and can simply enter your name and password on the Login page.

Plays Home Page

Now that you have been authenticated, you move on to the first secured page within the application, which is the home page. The Shakespeare's Plays Home Page appears as follows:

This page presents a list of options that allow the user to view all or a subset of plays, and an option to shop at the Globe Shop.

The following options are available to the Web user:

All	Displays all of Shakespeare's plays (from the plays table), *without* wrapping to the beginning of the list of plays when the end is reached.
All (Wrap to Beginning)	Displays all of the Shakespeare's plays, wrapping to the beginning of the list of plays when the end is reached.
By Type (Selector)	Displays a subset of Shakespeare's plays, based on the type selected using a selector control.
By Type (Hyperlink)	Displays a subset of Shakespeare's plays, based on the type selected using a hyperlink.
By Type (Graphical Hyperlink)	Displays a subset of Shakespeare's plays, based on the type selected using a graphical hyperlink.
Switch by Type (Graphical Hyperlink)	Displays a subset of Shakespeare's plays, based on the type selected using a SWITCH macro with a graphical hyperlink.

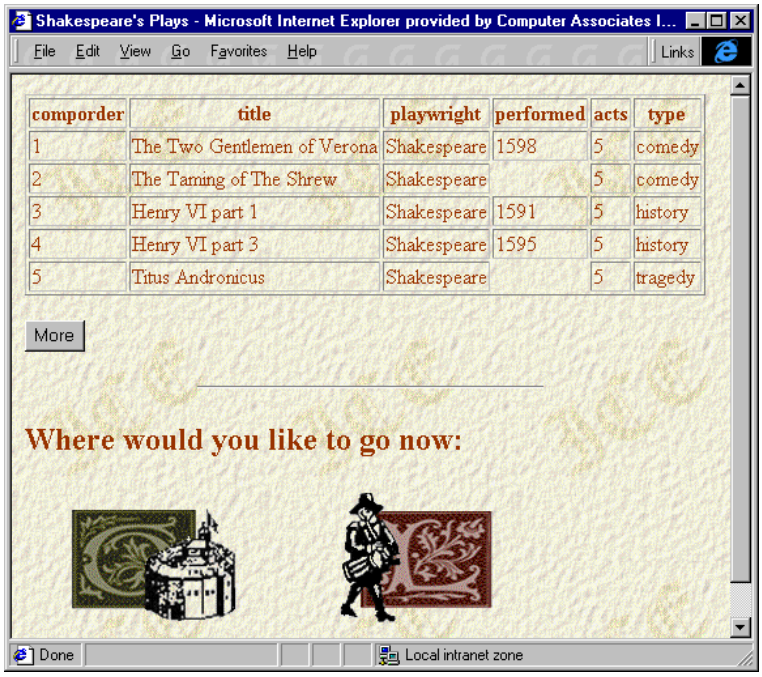
Note: This would be the desired result in an actual finalized application.

Plays View Options

There are several different methods of displaying data from the plays table in the icetutor database. Some of these methods are described in this section.

Viewing All Plays

If the All option is selected, all the plays in the plays table are presented in groups of five in a data browser, as follows:



The More button displays the next set of five plays in the browser until all the plays have been displayed. This option does not wrap to the beginning of the list, so that when the last play is displayed, the browser is empty.

This is not as desirable as if the first set of plays were to be displayed after the last play in the database. This is exactly the purpose of the second option, All (Wrap to Beginning), which *does* provide the wrap-around capability.

You can return to the home page at any time by clicking the left icon at the bottom of the page. Similarly, the right icon is clicked if a user wants to log out of the application.

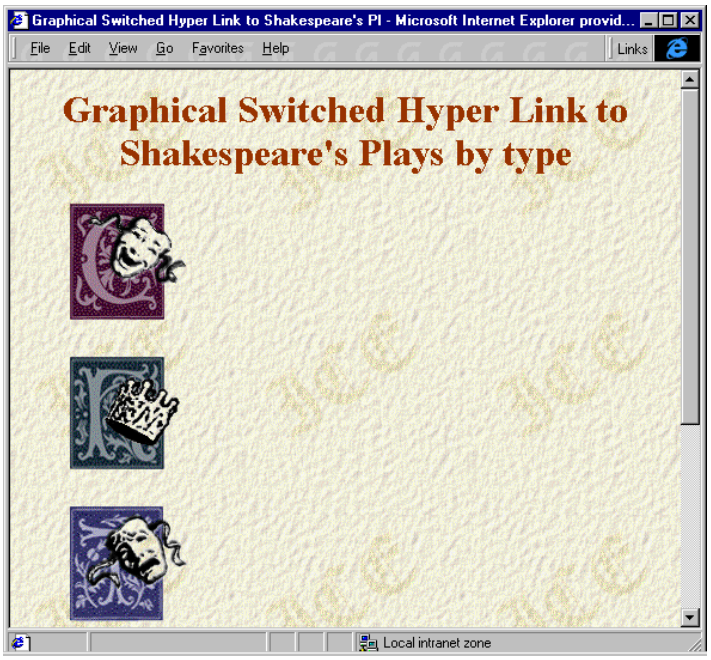
Viewing Selected Plays

The next four options on the home page menu allow a user to display a subset of the plays based on type. They each accomplish this in a different way. For example, the simplest example of this is evident in the By Type (Selector) option. A play type is chosen from a selector control, shown below:

After a play type is selected and Display clicked, a browser is displayed containing only plays of that type. Again, a More button allows the user to display more plays until the end of the list of plays is reached.

The other options show you how various implementations use hyperlinks for each of the play types in the database. The Switch by Type (Graphical Hyperlink) option is the programming example that we would like to highlight and the most desirable in terms of elegant Web page design and efficient HTML design.

This option produces the following Web page:



Here, the user can click the icon that represents the type of play they are interested in viewing. The results are again displayed in a browser with a More button.

Tip: You may want to take some time to familiarize yourself with play browsing portion of the application now before we move on to the next section describing the Globe Boutique. If you want to see how the browsing application was built, see Designing a Data Browsing Application in this chapter.

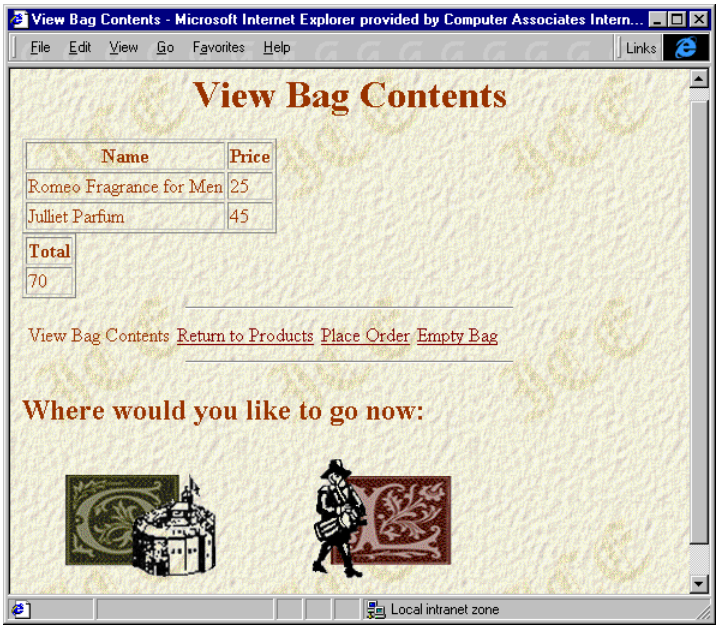
Globe Boutique

The Globe Shop option presents a home page for the Globe Boutique. On this page, a list of products that are available for purchase can be viewed and selected:

A user may see some interesting items and want to know more about them, so they would click the number of the item. A page that describes the item appears, providing more detail and its price. If the user is interested in purchasing the item, it can be easily added to their shopping bag (we doubt that shopping carts existed in Elizabethan England); otherwise, they can choose to go back to the product list.

Viewing Shopping Bag Contents

Once the item is added to the shopping bag, more items can be chosen or the contents of the shopping bag can be viewed, as shown below:



Each item in the shopping bag is listed, along with its price. Additionally, the total cost of the order is calculated and displayed at the bottom at the list.

Placing an Order or Emptying Shopping Bag

Notice that the available user options are hyperlinks that have been enabled (those that are not available are simply not enabled). The choices at this point are to add more items, empty the shopping bag, or place the order. If an order is placed and confirmed, a unique order number is assigned and that transaction is complete. If the bag is emptied, the items are removed and the user can start again.

We will now explore how the Plays application was created. You can actually perform the steps or just follow along, as desired.

> **Tip:** Take some time to familiarize yourself with the Globe Boutique application before we explore how the entire Plays application was created in the next section. If you would like to see how the code for the Globe Boutique portion of the application was developed, see Designing an Internet Shopping Application in this chapter.

Creating Application Directories

We will begin by creating the directories that we will need for our new My_Plays application. Two types of directories are needed—one that comes under Web Deployment Option security and one that does not.

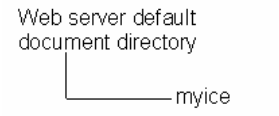
Creating Directories for Non-Web Deployment Option Registered Files

We have to create several directories under the Web root directory that will contain files that do not come under Web Deployment Option security and are visible to the HTTP server. These files include a welcome page HTML file and the facets it references, and a style sheet used with this page.

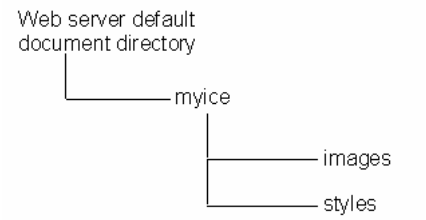
The welcome page for the my_plays application is the first page of the application. It must be accessible to any Web user because the user has not been authenticated yet. The welcome page is followed by the login page, which is public, but is resident in the plays business unit.

Create the directory structure for the non-Web Deployment Option files of My_Plays as follows:

a. Create a subdirectory under the Web server default document directory (for example, myice) that will hold the non-Web Deployment Option files for your application (that is, the welcome page):



b. Beneath this directory, create two other subdirectories that will hold the images and style sheet for the welcome page (for example, images and styles, respectively):



Note: The ice directory, and the images and styles subdirectories, were created for the Plays application at installation time. This can be used as a model for the myplays directory structure.

Creating Directories for Web Deployment Option-Registered Files

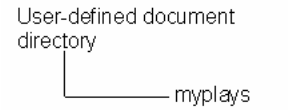
Next, an application directory for the majority of the files in your application has to be created. These files do come under Web Deployment Option security and instead of residing under the Web root directory, they will be created elsewhere in the file system. Later, we will register these files with the ICE server, which will then make the location visible through the HTTP server.

Create the directory structure for the Web Deployment Option files of My_Plays by performing the following steps:

1. In the desired (drive and/or) directory, create a directory for Web Deployment Option documents.

   For example, c:\ice\documents on Windows systems and /usr/web/documents on UNIX systems.

2. Create a subdirectory called myplays, which will contain all of the Web Deployment Option files for your application:



   Note: The plays directory beneath your Web Deployment Option directory was created for the Plays application at installation time. This can be used as a model for the directory structure on your machine (for example, the c:\ice\documents\myplays directory on Windows systems and /usr/web/documents/myplays on UNIX systems).

Creating Application Files

Next, we will proceed by creating the files that will comprise the My_Plays application. Again, there are files that are registered under Web Deployment Option and those that are not. In each case, pages, facets, and style sheets have to be created.

This section takes you through the process of creating or setting up these application components:

- An initial application page, not under Web Deployment Option control
- A welcome page and its referenced facets and style sheet, not under Web Deployment Option control
- A login page, under Web Deployment Option control
- The remainder of the Web Deployment Option-controlled application pages, facets, and style sheets

Creating the Starting Application Page

We first have to create the entry point for the application that is accessible through the Web server. In the Plays application, a single index file, ice_index.html, is the first file the Web user encounters. From this page, we will provide a hyperlink to the welcome page of the My_Plays application.

Note: You should also specify this initial file as the default document in your Web server setup or alternatively provide a link to it from your default document. Either way allows a Web user to simply enter an address of http://*machine_name* to access the initial page of the application. If you need more information, see the documentation supplied with your Web server.

To create the starting My_Plays application page:

1. Within the Web server default document directory, make a copy of the ice_index.html file, which was installed with Web Deployment Option, and rename the file "my_ice_index.html".

   For example, if you are using the Microsoft Internet Information Server, you might create:

   C:\InetPub\wwwroot\my_ice_index.html

2. In an HTML editor of your choice, modify the HTML in the new file, removing all options except the one to select the example tutorial application:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Style-Type"
    content="text/css">
<LINK HREF="/ice/styles/ice.css" type="text/css"
    REL="stylesheet">
<TITLE>Web Deployment Option</TITLE>
</HEAD>
<BODY>
<H1>Web Deployment Option</H1>
<P ID=W>Welcome to Web Deployment Option</P>
Choose the Following Option to Enter the My_Plays
    Application:
<P>
<TABLE>
<TR><TD><A HREF="/ice-bin/oiice.dll/my_plays[myplay_welcome.html">My
    Example Tutorial Application</A></TD></TR>
</TABLE>
</BODY>
</HTML>
```

3. Save your file and exit the HTML editor.

Creating the Welcome Page and Facets

For the welcome page, you will now create a skeleton file, in which you will enter a standard HTML template to be inserted into all the files eventually.

Creating the Welcome HTML File

To create the skeleton file for the welcome page file, perform the following steps:

1. In an HTML editor of your choice, enter the following code template. (Note that this code can be found in the play_welcome.html file, provided in the plays directory).

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0//EN">
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Style-Type"
    CONTENT="text/css">
<LINK HREF="<!-- #ICE
    INCLUDE=`my_plays[myplay_styleSheet.css]` -->"
    TYPE="text/css" REL="STYLESHEET">
<TITLE>Title of page</title>
</HEAD>
<BODY>
<H1>Title of page</H1>
</BODY>
</HTML>
```

2. Save the file with the name myplay_welcome.html in the myice subdirectory that you created, beneath the Web server root directory that you created. (For more information, see Creating Directories for Non-Web Deployment Option Registered Files in this chapter).
3. Check that you can read the HTML file with your browser by using File Open (or the equivalent).

   Note: The file will *not* be visible via your Web server.

Copying the Welcome Page Facets

Rather than creating new facets referenced by the welcome page for My_Plays, we will copy the facets from the Plays application:

1. From within your Web server default document directory, access the ice\images subdirectory on Windows, or ice/images subdirectory on UNIX.
2. Copy the bgpaper.gif and oldglobe.gif files to the myice\images subdirectory.

   For more information, see Creating Directories for Non-Web Deployment Option Registered Files in this chapter.
3. From within your Web server default document directory, access the ice\styles subdirectory.
4. Copy the ice.css file to the myice\styles subdirectory.

   For more information, see Creating Directories for Non-Web Deployment Option Registered Files in this chapter.

Creating the Remaining Pages and Facets

To save time, we will create skeleton files for all the pages we will need for our application at once, and then register them with the ICE server at the same time using Visual DBA. We will also copy the facets used in the Plays application, and later register with the new business unit.

Creating My_Plays Application Pages

To create the skeleton pages for the My_Plays application:

1. In an HTML editor of your choice, create a new HTML file named myplay_home.html, corresponding to the play_home.html original file in the plays directory.
2. Enter the code template found in the Creating the Welcome Page section. (Note that this code can also be found in the play_welcome.html file provided in the plays directory).
3. Save the file in your myplays application subdirectory.

   For more information, see Creating Directories for Non-Web Deployment Option Registered Files in this chapter.
4. Check that you can read the HTML file with your browser by using File Open (or the equivalent).

   Note: The file will *not* be visible via your Web server.
5. Repeat this procedure for all the files in the plays application directory, renaming them with the "my" prefix.

   The HTML file names are provided below for convenience:

myplay_all	myplay_shopDescribe
myplay_allWrap	myplay_shopHome
myplay_allWrapSub	myplay_shopRemove
myplay_autoUser	myplay_shopView
myplay_home	myplay_subSet
myplay_login	myplay_TxnCndCmt_h
myplay_newProduct	myplay_typeGLink
myplay_newProductInsert	myplay_typeGSLink
myplay_sessionControl_h	myplay_typeLink
myplay_shopAction_h	myplay_typeLinkSubSet
myplay_shopAdd	myplay_typeList
myplay_shopConfirm	

Copying the My_Plays Application Facets

We will now copy the image and style sheets files from the Plays application to the myplays directory.

1. From within your Ingres system directory, access the ingres\ice\plays subdirectory on Windows, or ingres/ice/plays subdirectory on UNIX.
2. Copy each of the .gif and .css files to the myplays subdirectory.

   For more information, see Creating Directories for Non-Web Deployment Option Registered Files in this chapter.

   The names of the images and style sheets to be used are shown below for your reference:

   - bgpaper.gif
   - comedy.gif
   - history.gif
   - logout.gif
   - oldglobe.gif
   - play_styleSheet.css
   - play_public.css
   - romance.gif
   - tragedy.gif

These files will be registered to a new business unit later in the tutorial for ease in maintenance and security purposes.

Using Style Sheets

The following code line includes a reference to the style sheet that is used for determining the styles that apply to the various elements in the application:

```
<LINK HREF="<!-- #ICE INCLUDE=
    `my_plays[myplay_styleSheet.css]` -->"
    TYPE="text/css"
    REL="STYLESHEET">
```

Style sheets help us to separate appearance or *style* from the information or *content* of our Web site. By including the style sheet using this technique, we are able to bring the style of our site under Web Deployment Option control in addition the content. In this tutorial, we use level one of the cascading style sheet mechanism, recommended by the World Wide Web Consortium (W3C) at the Web site http://www.w3.org/TR/REC-CSS1.

Gaining Access to Web Deployment Option Information

You use a Database Object Manager window within Visual DBA to access the objects on your ICE server. For information on how to start Visual DBA and access the Web Deployment Option information in the Database Object Manager, see "Chapter 4: Managing the Web Deployment Option."

The following sections take you through the management of objects using the ICE branches in the Database Object Manager.

Registering Your Files and Location

Through the ICE branch in the Database Object Manager in Visual DBA, you can set up security for your Web Deployment Option objects, and establish a server location for your files.

It is a requirement of the system that all objects to be made available through Web Deployment Option must be registered with a business unit. Each business unit must, in turn, be registered with a session group. We therefore begin by creating a new session group in this section.

Creating a Session Group

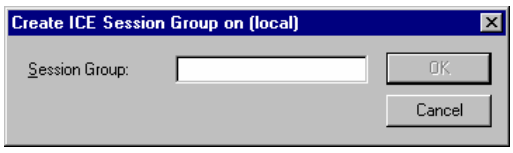
The first step in establishing security for our My_Plays application is the creation of a session group. This is used in the creation of cookies in the management of connections to the ICE server, in the case of a user opening more than one application within the same browser.

We will create the my_playgroup session group as follows:

1. Expand the ICE branch in the Database Object Manager.
2. Expand the Server branch.
3. Select the Session Groups branch.
4. Click the Add Object toolbar button.

Alternatively, choose Edit, Create.

The Create ICE Session Group Name dialog appears:



5. Enter **my_playgroup** in the Session Group edit control.
6. Click OK.

Setting Up Public Files

You must place those application files that need to be publicly accessible in the HTML root document directory or in an aliased directory. This includes the first file the user will encounter in the application, which is myplay_welcome.html. Move this file to the HTML root document directory for your Web server. For example, if using Microsoft Internet Explorer, this would be:

C:\InetPub\wwwroot

Next, we have to copy the graphic files that will appear in the welcome and login pages to the wwwroot\ice directory. These files include bgpaper.gif and oldglobe.gif.

The files that need to be public when being registered with Visual DBA are myplay_autoUser.html and myplay_login.html. These files include a page for user login and auto declaration—used if the user is not defined.

Creating a Server Location for Secured Pages

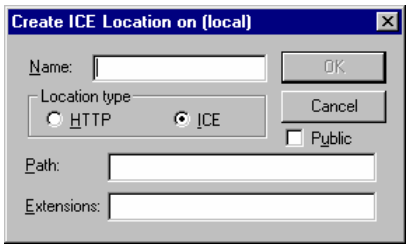
The next step is to create a server location, registering the application directory that you created earlier with the ICE server. We will later associate this location with a business unit.

To create the my_play_location location:

1. Expand the ICE branch in the Database Object Manager.
2. Expand the Server branch.
3. Select the Locations branch.
4. Click the Add Object toolbar button.

Alternatively, choose Edit, Create.

The Create ICE Location dialog appears:



5. In the Name edit control, enter the server name for the location, **my_play_location**.

   The ICE radio button should already be selected. Also, the Public check box should be cleared, which it is by default. (It indicates whether the location is available if no authentication of the user has been performed.)

6. In the Path edit control, enter the full path of your application subdirectory.

   This is the plays subdirectory under your chosen application directory on the local file system, discussed in the Creating Application Directories section.

7. Leave the Extensions edit control empty.

8. Click OK.

Creating a Business Unit

We need a business unit that is equivalent of plays for the Plays application. This business unit is a collection of HTML files, facets, and applications performing a similar or related function in our application.

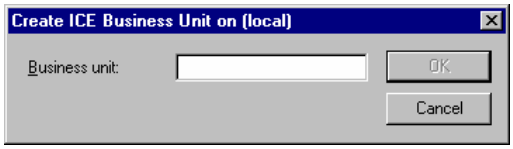
In a later section, you will see how you can associate a role with the business unit and grant permissions to it.

To create the my_plays business unit:

1. Expand the ICE branch in the Database Object Manager.
2. Select the Business Units branch.
3. Click the Add Object toolbar button.

   Alternatively, choose Edit, Create.

   The Create ICE Business Unit dialog appears:



4. Enter the name of the business unit, **my_plays**.
5. Click OK.

   Notice that if you expand the Business Units branch, the new my_plays business unit appears.

Associating the Server Location with the Business Unit

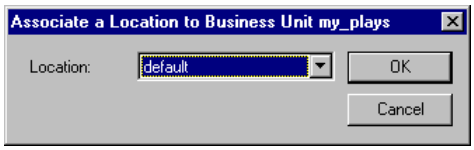
The next step is to create an association between the physical location of the application files and the business unit.

To associate the my_play_location server location with the my_plays business unit:

1. Expand the ICE branch in the Database Object Manager.
2. Expand the Business Units branch.
3. Expand the my_plays branch.
4. Select the Locations branch.
5. Click the Add Object toolbar button.

   Alternatively, choose Edit, Create.

   The Associate a Location to Business Unit dialog appears:



6. Select my_play_location from the drop-down list.
7. Click OK.

Associating Pages with the Business Unit

In order to group the pages for the My_Plays application together logically, we must associate them with the my_plays business unit.

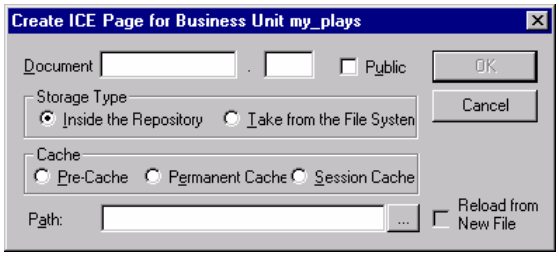
Note: You can use the regdocs utility to register multiple files simultaneously. For more information on this command, see the *Command Reference Guide*.

To associate a page with the my_plays business unit:

1. Expand the ICE branch in the Database Object Manager.
2. Expand the Business Units branch.
3. Expand the my_plays business unit branch.
4. Select the Pages branch.
5. Click the Add Object toolbar button.

   Alternatively, choose Edit, Create.

The Create ICE Page for Business Unit dialog appears:

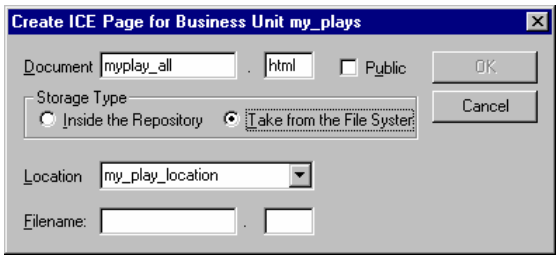


6. In the Document edit controls, enter the name and extension, respectively, of the document file (for example, **myplay_all** and **html**).

   Note: For this example, we are choosing to use the same name for the document and the actual HTML page.

7. Select the Take from File System option.

   The dialog changes as follows:



   Notice that the my_play_location location is already selected.

8. In the Filename edit controls, enter the name and extension, respectively, of the HTML file (for example, **myplay_all** and **html**).
9. Click OK.
10. Repeat steps 5–9 for each of the HTML files in your myplays application directory.

Associating Facets with the Business Unit

You must also include the facets together with the pages logically, requiring us to associate them with the my_plays business unit.

Note: You can use the regdocs utility to register multiple files simultaneously. For more information on this command, see the *Command Reference Guide*.

To associate a facet with the my_plays business unit:

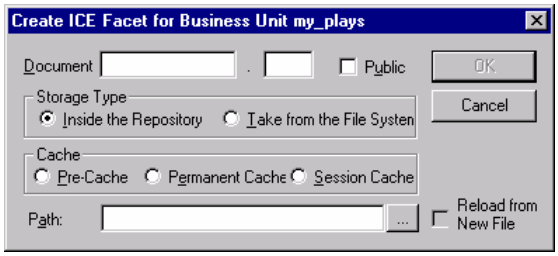
1. Expand the ICE branch in the Database Object Manager.
2. Expand the Business Units branch.
3. Expand the my_plays business unit branch.

4. Select the Facets branch.

5. Click the Add Object toolbar button.

   Alternatively, choose Edit, Create.

   The Create ICE Facet for Business Unit dialog appears:

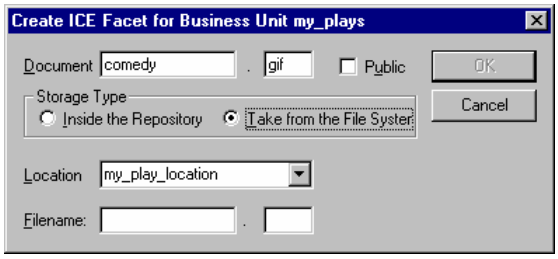


6. In the Document edit controls, enter the image file name and extension, respectively (for example, **comedy** and **gif**).

   Note: For this example, we are choosing to use the same name for the document and the actual facet.

7. Select the Take from File System option.

   The dialog changes as follows:



   Notice that the my_play_location location is already selected.

8. In the Filename edit controls, enter the name and extension, respectively, of the HTML file (for example, **comedy** and **gif**).

9. Click OK.

10. Repeat steps 5–9 for each of the graphic and style sheet files in your myplays application directory. For the following files, also select the Public check box:

    - bgpaper.gif
    - oldglobe.gif
    - play_public.css

    Note: Making these files public allows them to be accessed by any unauthenticated user while in the application.

Creating a Database Connection

To create an alias for the icetutor database and the user that owns the Ingres installation, the my_play_database database connection will be created.

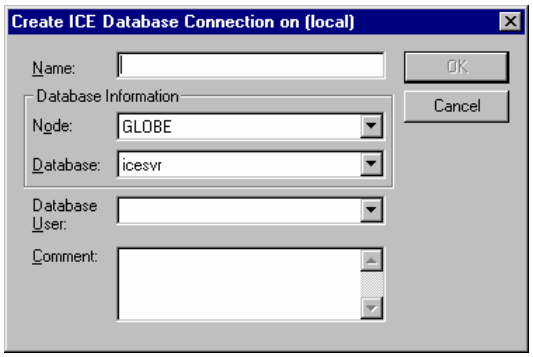
Note: This database connection can then be associated with a Web user using the Associate DB Connection to Web User dialog. This is left as an exercise for the reader.

To create the my_play_database database connection:

1. Expand the ICE branch in the Database Object Manager.
2. Expand the Security branch.
3. Select the Database Connections branch.
4. Click the Add Object toolbar button.

   Alternatively, choose Edit, Create.

   The Create ICE Database Connection dialog appears:



5. Enter **my_play_database** in the Name edit control.
6. Select icetutor from the Database drop-down list.
7. Select icedefdb from the Database User drop-down list.
8. In the Comment edit control, enter the following text:

   Database dedicated to the works of the Bard of Stratford-Upon-Avon

9. Click OK.

Creating a Profile

The security administrator of the Web Deployment Option Web site will want to create a profile that defines the general capabilities of a user that logs in using the auto-declaration page.

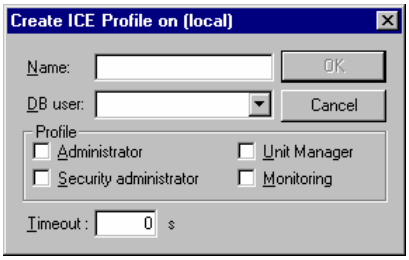
Next, we want to create a profile that can be assigned to a default user when a Web user declares an account for themselves.

To create the my_play_profile profile:

1. Expand the ICE branch in the Database Object Manager.
2. Expand the Security branch.
3. Select the Profiles branch.
4. Click the Add Object toolbar button.

   Alternatively, choose Edit, Create.

   The Create ICE Profile dialog appears:



5. Enter **my_play_profile** in the Name edit control.
6. Select icedefdb from the DB User drop-down list.

   All other edit controls will be left blank. They can be modified later, if necessary, by altering the profile.
7. Specify **300** seconds in the Timeout edit control.
8. Click OK.

Creating a Role

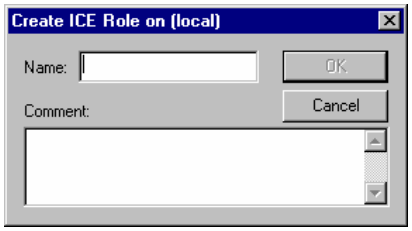
A role definition is needed so that we can associate the my_play_profile profile and the my_plays business unit with it, as you will see in the sections that follow.

To create the my_play_role role:

1. Expand the ICE branch in the Database Object Manager.
2. Expand the Security branch.
3. Select the Roles branch.
4. Click the Add Object toolbar button.

   Alternatively, choose Edit, Create.

The Create ICE Role dialog appears:



5. In the Name edit control, enter **my_play_role**.
6. In the Comment edit control, enter Role for the My_Plays Web application.
7. Click OK.

Associating a Role with a Profile

Now that we have a role created, we can associate the my_play_profile profile with it. The my_play_role will eventually have the Execute Documents permission granted to it.

To associate the my_play_role role with the my_play_profile profile:

1. Expand the ICE branch in the Database Object Manager.
2. Expand the Security branch.
3. Expand the Profiles branch.
4. Expand the my_play_profile branch.
5. Select the Roles branch.
6. Click the Add Object toolbar button.

Alternatively, choose Edit, Create.

The Associate Role to ICE Profile dialog appears:

7. Select my_play_role from the Role drop-down list.
8. Click OK.

Associating a Database Connection with a Profile

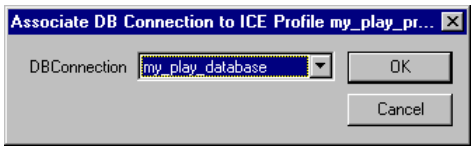
We also want to associate a database connection with the profile we have created. This associates the profile with the icetutor database and icedbuser database user.

To associate the my_play_database database connection with the my_play_profile profile:

1. Expand the ICE branch in the Database Object Manager.
2. Expand the Security branch.
3. Expand the Profiles branch.
4. Expand the my_play_profile profile branch.
5. Select the Database Connections branch.
6. Click the Add Object toolbar button.

   Alternatively, choose Edit, Create.

   The Associate DB Connection to ICE Profile dialog appears:



7. Select my_play_database from the DBConnection drop-down list.
8. Click OK.

Associating a Role with a Business Unit

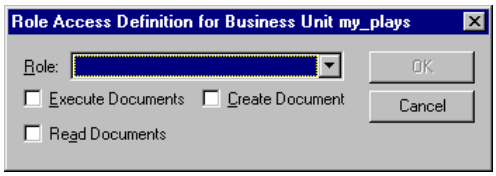
A Web Deployment Option role allows a group of users to be granted appropriate access rights by business unit owners to their business unit as a whole or on a per-page or per-facet basis.

We will now associate the my_play_role role with the my_plays business unit, granting the Execute Documents permission:

1. Expand the ICE branch in the Database Object Manager.
2. Expand the Business Units branch.
3. Expand the my_plays branch.
4. Expand the Security branch.
5. Select the Roles branch.
6. Click the Add Object toolbar button.

   Alternatively, choose Edit, Create.

The Role Access Definition for Business Unit dialog appears:



7. Select my_play_role from the Role drop-down list.
8. Select the Execute Documents check box.
9. Click OK.

Designing a Data Browsing Application

Now that the security objects for our application are created, we can go on to create the HTML pages that will make up the application. You can use the skeleton files that we created previously, as discussed in the Creating Application Files section.

We will examine the code used in the Plays application provided with Web Deployment Option for the viewing of Shakespeare's plays. In creating the My_Plays application, you will be instructed to add the code that is shown to your new files or copy it from an original Plays file.

Tip: Throughout this chapter, you can use the Plays HTML files to extract the code and paste it into the new file, so that you don't have to retype the code. However, you may want to reproduce the code manually as an exercise in using the Web Deployment Option macros.

Creating a Welcome Page

The welcome page is the first page that the user encounters. It lies outside the control of the ICE server and is a standard HTML page that does not come under the control of Web Deployment Option.

The welcome page is needed to provide access to the pages that *do* come under the control of the ICE server. It contains some welcome text and a link to the login page and therefore must be accessible through your Web server.

Important! The welcome page of your Web application should be accessible through the Web server.

Welcoming the User to Your Site

An example of a welcome page is the play_welcome.html file, which will be reproduced in our new file. Add the following code to the myplay_welcome.html file:

```
<HTML>
<HEAD>
<TITLE>
Shakespeare Live
</TITLE>
</HEAD>
<BODY>
<CENTER>
<H1>
Shakespeare Live
</H1>
</CENTER>
<CENTER>
Welcome to the <B>globe</B>
site for Shakespeare Studies.
<P>
<A HREF=/ice-bin/oiice.dll/
    my_plays[myplay_login.html]>
    <IMG SRC="/myice/images/oldglobe.gif" alt="Enter the Globe Here"></A>
</CENTER>
</BODY>
</HTML>
```

The line of interest here is the anchor:

```
<A HREF=/ice-bin/oiice.dll/
    my_plays[myplay_login.html]>
    <IMG SRC="/myice/images/oldglobe.gif" alt="Enter the Globe Here"></A>
```

This instructs the ICE server to access the myplay_login page, which resides in the my_plays business unit. The square brackets in the syntax show that the page is part of the business unit.

The oldglobe.gif file is visible to the HTTP server and resides under the ice/images directory beneath the root document directory within your Web server directory.

Note: There is no support for session groups at this level because we have not yet logged in and therefore have not yet been assigned a session ID.

The next page to be created is the login page.

Creating a Login Page

It is necessary to log in to the ICE server before you can access any pages held under its control. An example of a login page is the play_login.html page, which we will use to create myplay_login.html. It allows Web users to create an account for themselves by leaving the entry fields blank.

Establishing a Connection to the Server

Add the following code to the myplay_login.html file:

```
 <HTML>
 <HEAD>
 <TITLE>Shakespeare Plays </TITLE>

</HEAD>
<BODY>
<CENTER>
<H1>Shakespeare Plays </H1></TD>
</CENTER>
Thank you for visiting the Globe Centre for Shakespeare.
<P>
If you are a new visitor to our site, leave the
user name and password fields <I>blank.</I>
<BR>
Click on the
picture of the original Globe below from whence you
will be able to register with us.
<P>
If you already have an account, please fill in
your user name and password below.
<BR>
Click on the
picture of the original Globe below.
<P>
We hope you enjoy your visit!
<HR>
<FORM ACTION="/ice-bin/oiice.dll/
    my_plays[myplay_home.html]" METHOD="POST">
<INPUT TYPE=hidden NAME="ii_action" value="connect">
<INPUT TYPE=hidden
    NAME="ii_error_url" value=
    "/my_plays[myplay_autoUser.html]">
<CENTER>
<TABLE BORDER=0 ALIGN=CENTER VALIGN=CENTER>
    <TR>
    <TD>Please enter your:</TD>
    </TR>
    <TR>
    <TD>Name: </TD>
    <TD><INPUT SIZE=32 NAME="ii_userid"></TD>
    </TR>
    <TR>
    <TD>Password: </TD>
    <TD><INPUT SIZE=32 TYPE=PASSWORD
        NAME="ii_password"></TD>
    </TR>
</TABLE>
<INPUT TYPE="IMAGE" BORDER=0 NAME="connect"
    SRC="/myice/images/oldglobe.gif" ALT="Press Here to Enter the Globe Experience">
</CENTER>
</FORM>
</BODY>
</HTML>
```

This page sets the value of the hidden variable ii_action to "connect," collects the account name and password from the user, and passes them all to the my_plays[myplay_home.html] page when the user clicks Connect:

```
<FORM ACTION="/ice-bin/oiice.dll/
    my_plays[myplay_home.html]" METHOD="POST">
<INPUT TYPE=hidden NAME="ii_action" value="connect">
...
<INPUT TYPE="IMAGE" BORDER=0 NAME="connect"
    SRC="/ice/images/oldglobe.gif" ALT="Connect">
```

If the user has no account, the login will fail and the failure action (specified by ii_error_url) will take them to the page my_plays[myplay_autoUser.html]. This page allows the user to create a new account. We will look at how to do that later. First, we will take a quick look at the home page for our system.

Creating a Home Page

Once the Web user has logged in, they are presented with a home page, which instructs them of the options that are available to them. You will notice the introduction of a session group to the HTML code for this page.

For our home page, we will demonstrate the INCLUDE feature. This allows the Web author to incorporate generic Web Deployment Option code on many pages. An example of its use would be to include code to ensure that each page had a uniform style across the site. In our case, we use the INCLUDE file for another reason; that of including Web Deployment Option code to ensure any open transactions are closed. Little else is new other than a list of links to the various pages that demonstrate various HTML or Web Deployment Option features.

Using INCLUDE file to Commit Open Transactions

Add the following code to the myplay_home.html page:

```
 <HTML>

 <HEAD>

 <TITLE>Shakespeare's Plays Home Page</TITLE>

</HEAD>
<BODY>
<CENTER>
<H1>Shakespeare's Plays Home Page</H1>
</CENTER>
This is the home page for Shakespeare's plays hosted on the server globe
<H2>
View Shakespeare's Plays
</H2>
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_TxnCndCmt_h.html]` -->
```

```
<OL>
<LI> <A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_all.html]">All</A>
<LI> <A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_allWrap.html]">
    All (wrap to beginning)</A>
<LI> <A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeList.html]">By type (selector)</A>
<LI> <A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLink.html]">
    By type (hyper-link)</A>
<LI> <A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeGLink.html]">
    By type (Graphical hyper-link)</A>
<LI> <A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeGSLink.html]">
    Switch by type (Graphical hyper-link)</A>
<LI> <A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_newProduct.html]">Add a product</A>
</OL>
<A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_shopHome.html]">Globe
    Shop</A>
<P>
Please
<A HREF="/ice-bin/oiice.dll/
    my_plays[myplay_login.html]?ii_action=disconnect">
    logout</A>,don't time out!
</BODY>
</HTML>
```

The line that includes the extra code is:

```
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_TxnCndCmt_h.html]` -->
```

The syntax for specifying an include file is the same as for when one is linked to. Here the business unit is my_plays and the file name is myplay_TxnCndCmt_h.html. We need to specify the REPEAT keyword because there are Web Deployment Option macro statements to be evaluated in this file.

In the next section, you will create a page to create a new account.

Creating a User Account Automatically

Anyone accessing non-public pages in a Web Deployment Option-controlled part of a Web site must have an account. Since this could mean creating accounts for a large number of Web users, there needs to be a way for the users to allocate themselves an account. The account should have the minimum permissions required, by associating a role using Visual DBA.

Adding an Auto-Declaration Page

The following is an example of an auto-declaration page. This code should be added to the myplay_autoUser.html file:

```
</HEAD>
<BODY>
<H1>Automatic Declaration</H1>
<FORM ACTION="/ice-bin/oiice.dll/
    my_plays[myplay_home.html]" METHOD="POST">
<INPUT TYPE=hidden NAME="ii_action" value="declare">
<INPUT TYPE=hidden NAME="ii_profile"
    value="myplay_profile">

<CENTER>
<TABLE BORDER=0 ALIGN=CENTER VALIGN=CENTER>
    <TR>
    <TD>Please enter your:</TD>
    </TR>
    <TR>
    <TD>Name: </TD>
    <TD><INPUT SIZE=32 NAME="ii_userid"></TD>
    </TR>
    <TR>
    <TD>Password: </TD>
    <TD><INPUT SIZE=32 TYPE=PASSWORD
        NAME="ii_password"></TD>
    </TR>
</TABLE>
<INPUT TYPE="IMAGE" BORDER=0 NAME="connect"
    SRC="/ice/images/oldglobe.gif" ALT="Connect">
</CENTER>
</FORM>
</BODY>
</HTML>
```

The action for the form is to execute the myplay_home.html document in the my_plays business unit. It is passed, along with the new user name and password, the declare action. This causes a new Web user to be created with the my_play_profile profile.

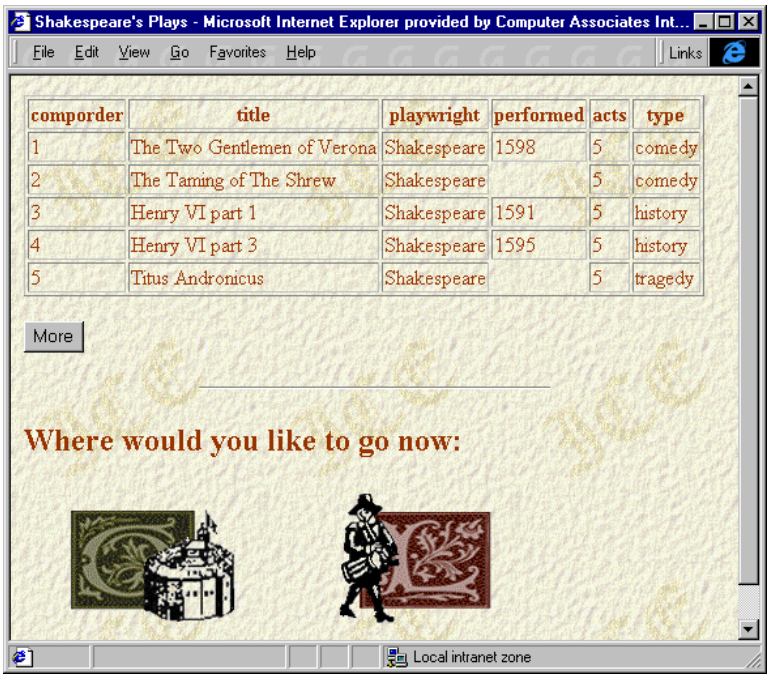
Note: The my_plays profile must have been created previously using the Create ICE Profile dialog in Visual DBA. See Creating a Profile in this chapter.

Displaying All Table Rows

In the first Web Deployment Option feature programming example, we will show how to specify an SQL Web Deployment Option macro keyword and then embed the macro into a simple Web Deployment Option document. We will also take advantage of the CURSOR macro keyword to add a More button to the document, enabling us to view the sequence of plays by subset.

Finally, we will arrange to commit the transaction upon returning to the home page. The code will be placed into a separate file and included into the home page and the myplay_all.html document using the INCLUDE keyword. Communication of the transaction name will be achieved with Web Deployment Option session variables.

The page used to display all the rows in the plays table is shown below:



Using a Simple Select Statement

We would like to present the data in the plays table in the icetutor database, five rows at a time in an HTML tabular format. A transaction name and cursor name will also be established.

Constructing Macro

The first task is to construct a Web Deployment Option macro using the SQL keyword, as follows:

```
<!-- #ICE
    DATABASE = `icetutor`
    SQL=`select * from plays`
    TRANSACTION=`Complete`
    CURSOR=`Works`
    ROWS=`5`
    TYPE=`TABLE`
-->
```

If we were to display this page in a browser, the output would appear as follows:

comporder	title	playwright	performed	acts	type
1	Two Gentlemen of Verona	Shakespeare	1598	5	comedy
2	Taming of The Shrew	Shakespeare		5	comedy
3	Henry VI part 1	Shakespeare	1591	5	history
4	Henry VI part 3	Shakespeare	1595	5	history
5	Titus Andronicus	Shakespeare		5	tragedy

Adding the Macro to Your HTML

We are happy—for now—with this output, so we cut and paste the macro text into the "myplay_all.html" file. The result is as follows:

```
<HTML>
<HEAD>
<TITLE>Shakespeare's Plays</TITLE> </HEAD>
<BODY>
<H1>Shakespeare's Plays</H1>

<!-- #ICE DATABASE = `icetutor`
    SQL=`select * from plays`
    TRANSACTION=`Complete`
    CURSOR=`Works`
    ROWS=`5`
    TYPE=`TABLE` -->

</BODY>
</HTML>
```

Viewing the Page

Since you have already registered the file, you can now view the page by logging in to your site and specifying its address:

Windows

http://*your_machine_name*/ice-bin/oiice.dll/my_playgroup/my_plays[myplay_all.html]

UNIX

http://*your_machine_name*/ice-bin/oiice.1.so/my_playgroup/my_plays[myplay_all.html]

Notice that the only way of retrieving the next set of rows is by reloading the page (by clicking the Refresh or Reload button in your browser). In the next section, we will add a More button to do this more elegantly.

Adding a More Button

We now need a more efficient way of retrieving the next set of five plays. Since we have opened both a transaction (Complete) and a cursor (Works)—and the action of the cursor is to retrieve the next set of rows—all we need to do is to reload the document. You might like to try this now with your document.

Reloading the Browser with a Submit (More) Button

Although this will work, it is not quite as user-friendly as we would like! It is preferable to add a button to get more plays. The action of the button is to revisit the document. If you have read the "HTML Primer" appendix, you know that a button exists on a form so we need a form whose action is the address of the *current* document. The button type that submits the form to the Web server is Submit.

The result of all of this is that the Submit button acts just like the Refresh or Reload button on your browser. We assign a meaningful name to the button (such as "More"). We are now ready to present the myplay_all.html document with our new "More" button.

Adding the More Button to the Form

Update the myplay_all.html file to include the following code:

```
<HTML>
<HEAD>
<TITLE>Shakespeare's Plays</TITLE>
</HEAD>
<BODY>
<H1>Shakespeare's Plays</H1>

<!-- #ICE
        DATABASE = `icetutor`
        SQL=`select * from plays`
        TRANSACTION=`Complete`
        CURSOR=`Works`
        ROWS=`5`
        TYPE=`TABLE`
        -->

<P>
<FORM ACTION="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_all.html]" METHOD="POST">
<INPUT TYPE="submit" NAME="More" VALUE="More"
    ALT="Show more plays">
</FORM>
</BODY>
</HTML>
```

Including Generic Session Control

We would obviously like to give the user some way of returning to the home page, or logging out. We can easily do this with the following macro statements:

```
<HR WIDTH="50%" >
<H2>Where would you like to go now:</H2>
<TABLE BORDER=0 CELLSPACING=3>
<TR>
<TD>
<A HREF="/ice-bin/oiice.dll/my_plays[myplay_home.html]">Globe Home Page:</A>
<TD>
<A HREF="/ice-bin/oiice.dll/my_plays[myplay_home.html]">
    <IMG SRC="/myice/images/oldglobe.gif" alt="Return to Globe Home Page"
</A>
<TR>
<TD>
<P>
<A HREF="/ice-bin/oiice.dll/my_plays[myplay_login.html]?ii_action=disconnect">
    logout:<A>
<TD>
<A HREF="/ice-bin/oiice.dll/my_plays[myplay_login.html]?ii_action=disconnect">
    <IMG SRC="/ice-bin/oiice.dll/my_plays[logout.gif]" alt="Logout">
</A>
</TABLE>
<HR>
```

Clearly, we could insert this directly into our file; then every other one that we create. Then, revisit every file whenever we wish to change something. You get the idea. It would be much easier to have one version of this in a separate file and include it as needed. To do this, we save the code in the file named myplay_SessionControl_h.html and add the INCLUDE line to the myplay_all.html file.

Note: Any INCLUDE files need to be registered (that is, associated with a business unit) with the ICE server using the Create ICE Page for Business Unit dialog. (You should have already registered the file in the Associating Pages with the Business Unit section).

Using INCLUDE file to Control Sessions

This code adds the INCLUDE line to the myplay_all.html file:

```
<HTML>
<HEAD>
<TITLE>Shakespeare's Plays</TITLE>
</HEAD>
<BODY>
<H1>Shakespeare's Plays</H1>
<!-- #ICE
        DATABASE = `icetutor`
        SQL=`select * from plays`
        TRANSACTION=`Complete`
        CURSOR=`Works`
        ROWS=`5`
        TYPE=`TABLE`
        -->
<P>
<FORM ACTION="/ice-bin/oiice.dll/
    my_playgroup/my_plays[myplay_all.html]" METHOD="POST">
```

```
<INPUT TYPE="submit" NAME="More"         VALUE="More"
    ALT="Show more plays">
</FORM>
<!-- #ICE REPEAT
    INCLUDE=`my_plays[myplay_SessionControl_h.html]`
-->
</BODY>
</HTML>
```

Adding Transaction Control

The document as it stands opens a transaction named "Complete," but never closes it with a commit or rollback. We obviously cannot commit the transaction on the same page because then we would only ever retrieve the first result subset. We could create an extra page, which we visit for the sole purpose of committing the transaction, but we effectively already have such a page—our home page. A good time to commit the transaction would be when we transfer to the home page.

There could be many transactions that use this mechanism, so we choose a variable to contain the name of the transaction to be committed. Another variable will record the fact that the transaction is either available to be committed or has already been committed (since it is the nature of HTML/HTTP that we have no control over how the user arrives at the home page or how often).

Declaring Variables to Control Transactions

Add the two DECLARE macros shown below to your myplay_all.html file. This is the final version of the file, with transaction control information:

```
<HTML>
<HEAD>
<TITLE>Shakespeare's Plays</TITLE>
</HEAD>
<BODY>
<H1>Shakespeare's Plays</H1>
<!-- #ICE
        DATABASE = `icetutor`
        SQL=`select * from plays`
        TRANSACTION=`Complete`
        CURSOR=`Works`
        ROWS=`5`
        TYPE=`TABLE`
        -->
<!-- #ICE DECLARE=`session.e_playTxn=Complete` -->
<!-- #ICE DECLARE=`session.e_playTxnCommitted=FALSE`
-->
<P>
<FORM ACTION="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_all.html]" METHOD="POST">
<INPUT TYPE="submit" NAME="More" VALUE="More"
    ALT="Show more plays">
</FORM>
<!-- #ICE REPEAT
    INCLUDE=`my_plays[myplay_SessionControl_h.html]`
-->
</BODY>
</HTML>
```

Committing Transactions on the Home Page

There are various variable scopes available to us: server, session, and page. The naming convention used for variables in this tutorial is *scope_name*, where *scope* is represented by one of the following prefixes:

Scope Prefix	Scope Level
s	Server
e	sEssion
p	Page

This means that the variable e_playTxnCommitted is a *session*-level variable.

Using Variables to Commit Transactions

We now need some code to use these variables to commit the transaction at the appropriate time. The code that achieves this conditional transaction commit appears below. It should be added to the myplay_TxnCndCmt_h.html file:

```
<!-- #ICE REPEAT IF (DEFINED (e_playTxnCommitted))
    THEN=``
    ELSE=`<!-- #ICE
            DECLARE=``session.e_playTxnCommitted=TRUE``
            -->`
-->

<!-- #ICE REPEAT IF (DEFINED(e_playTxn) AND
    (`:e_playTxnCommitted` != `TRUE`))
    THEN=`<!-- #ICE COMMIT=``:e_playTxn`` -->
    <!-- #ICE
        DECLARE=``session.e_playTxnCommitted=TRUE``
    -->`
-->
```

Perform Conditional Transaction Committals

We add an IF macro statement to our next document; suffice it to say for now that the first part of this code tests if the e_playTxnCommitted variable exists. If it does not, it is created and set to the value TRUE. We need to do this because the first time we visit the home page, no variables will be set. Then, if the e_playTxn variable exists and the transaction has *not* yet been committed, it is committed. We need to perform these checks because it is an error to commit a non-existent transaction.

It only remains to include this file in our home page, myplay_home.html, as follows:

```
<HTML>
<HEAD>
<TITLE>Shakespeare's Plays Home Page</TITLE>
</HEAD>
<BODY>
<CENTER>
<H1>Shakespeare's Plays Home Page</H1>
</CENTER>
This is the home page for Shakespeare's plays hosted on the server <B>globe</B>
<H2>
View Shakespeare's Plays
</H2>

<!-- #ICE REPEAT
    INCLUDE=`my_plays[myplay_TxnCndCmt_h.html]` -->

<OL>
<LI> <A HREF="/ice-bin/oiice.dll/my_playgroup
    /my_plays[myplay_all.html]">All</A>
</OL>
Please
<A HREF="/ice-bin/oiice.dll
    /my_plays[myplay_login.html]?ii_action=
    disconnect">logout</A>, don't time out!
</BODY>
</HTML>
```

The next section illustrates the use of the IF keyword.

Displaying All Table Rows with Wrapping

In the previous example, when the information in the table was exhausted, the page displayed the column headers and no rows. If you have not seen this, you might like to try it now by clicking More until there are no more rows to display. (*Hint: The number of rows returned is available in the ii_rowcount variable*.)

Testing for End of Result Set

We would prefer the user to be presented with a way of resetting the cursor to the beginning of the result set again, to be able to return to the beginning. We choose to do this only when the number of rows returned is not 5 (the requested number). When there are more than five rows available, the ICE server will always return five and set ii_rowcount accordingly. When there are fewer than five rows to return, ii_rowcount will be set to the appropriate value. We can, therefore, test the inequality of ii_rowcount with 5.

The select statement is as before but now we include the test. If we receive fewer than five rows (ii_rowcount != 5), we insert the HTML to visit another page which is a facsimile of this page, but commits the transaction before running exactly the same query. This allows us to restart from the beginning of the result set.

Adding an IF Statement to Wrap to Beginning of Result Set

Our solution should be added to the myplay_allWrap.html file, as follows:

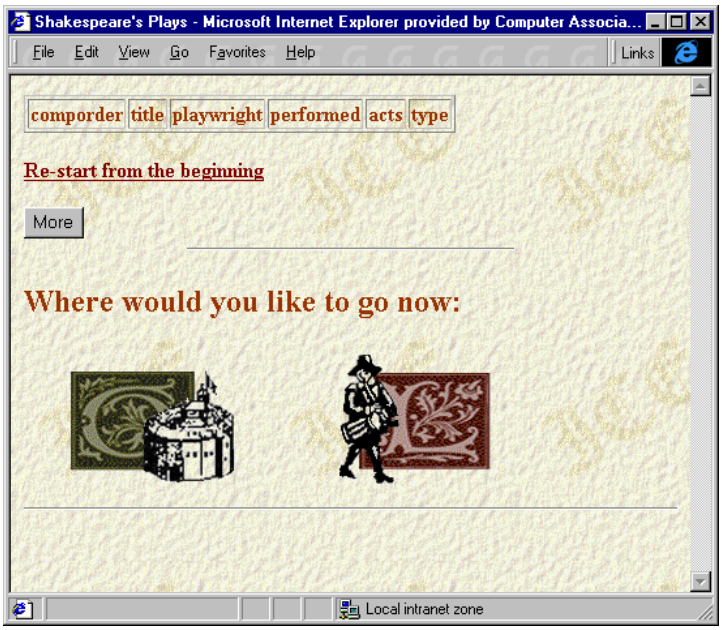
```
 <HTML>
 <HEAD>
 <TITLE>Shakespeare's Plays</TITLE>

</HEAD>
<BODY>
<FORM ACTION="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_allWrap.html]" METHOD="POST">

<!-- #ICE REPEAT
        DATABASE = `icetutor`
        SQL=`select * from plays`
        TRANSACTION=`Complete`
        CURSOR=`Works`
        ROWS=`5`
        TYPE=`TABLE`
-->

<!-- #ICE IF (`:ii_rowcount` != `5`)
    THEN=`<P>
    <B>
    <A HREF="/ice-bin/oiice.dll/my_playgroup/
        my_plays[myplay_allWrapSub.html]">
        Re-start from the beginning
        </A></B><P>`
-->

<!-- #ICE DECLARE=`session.e_playTxn=Complete` -->
<!-- #ICE DECLARE=`session.e_playTxnCommitted=FALSE` -->
<P>
<INPUT TYPE="submit" NAME="More" VALUE="More"
    ALT="Show more plays">
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->
</FORM>
</BODY>
</HTML>
```

The interesting part of the file is:

```
<!-- #ICE IF (`:ii_rowcount` != `5`)
    THEN=`<P>
    <B>
    <A HREF="/ice-bin/oiice.dll/my_playgroup/
        my_plays[myplay_allWrapSub.html]">
        Re-start from the beginning
        </A></B><P>`
-->
```

If the number of rows returned is *not* 5, it must be less than 5. We must test the inequality here because the IF statement performs *string* comparisons. If the number of rows is less than 5 (we have reached the end of the result set), we wish to give the user the opportunity to visit a "reset" page. The THEN branch of the IF inserts the required link.

The page will appear as follows once the user has reached the end of the plays in the browser:



Committing Previous Transaction and Starting Transaction Again

In order to start again, the first thing we need to do is to commit the previous transaction:

```
<!-- #ICE COMMIT=`:e_playTxn` -->
```

We then start *exactly the same* transaction again, giving the illusion that the user has visited the same file. Notice how the link to retrieve MORE entries returns the user to the main page. This page is therefore only used to commit and restart the transaction. This technique can also be used to ensure that a result set is current.

Let us take a look at the code below. You should add this code to the next file, named myplay_allWrapSub.html:

```
<HTML>
<HEAD>
<TITLE>Shakespeare's Plays</TITLE>
</HEAD>
<BODY>

<!-- #ICE COMMIT=`:e_playTxn` -->

<!-- #ICE
        DATABASE = `icetutor`
        SQL=`select * from plays`
        TRANSACTION=`Complete`
        CURSOR=`Works`
        ROWS=`5`
        TYPE=`TABLE`
-->
```

```
<!-- #ICE DECLARE=`session.e_playTxn=Complete` -->
<!-- #ICE DECLARE=`session.e_playTxnCommitted=FALSE`
-->

<FORM ACTION="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_allWrap.html]" METHOD="POST">
<INPUT TYPE="submit" NAME="More" VALUE="More"
    ALT="Show more plays">
</FORM>

<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->
</BODY>
</HTML>
```

Notice how the link to retrieve 'MORE' entries returns the user to the main page. This page is therefore only used to commit and restart the transaction. This technique can also be used to ensure that a result set is current.

The next section shows how Web Deployment Option can generate the tags needed for a selector control.

Creating an Automatically-Generated Selector Control

We have seen how to retrieve a result set from the database. Now we would like to retrieve the plays according to type. We note that for the purposes of these examples, the schema is somewhat denormalized. Normalization and the subsequent adjustment of the queries are left as an exercise for the reader.

One way of presenting Web users with a choice is to use a selector control. Web Deployment Option automatically generates the necessary HTML tags to generate a selector control element for us when we specify the SELECTOR keyword for an SQL query. We must supply a variable name to contain the value selected by the user and we specify this by using the ATTR keyword to name the e_type variable type.

The following page uses a selector control and a Display button:

Using a Selector Control to Obtain Play Type

The following code should be added to the myplay_typeList.html page, as shown below:

```
<HTML>
<HEAD>
<TITLE>Select a Type: Shakespeare's Plays</TITLE>
</HEAD>
<BODY>
<H1>Select a Type: Shakespeare's Plays</H1>
<FORM ACTION="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_subSet.html]" METHOD="POST">

<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_TxnCndCmt_h.html]` -->

<!-- #ICE DECLARE=`session.e_type='select play type'` -->

<!-- #ICE REPEAT
    DATABASE = `icetutor`
    SQL=`select distinct type from plays`
    TRANSACTION=`t_typeList`
    CURSOR=`Works`
    ROWS=`10`
    TYPE=`SELECTOR`
    ATTR=`NAME=e_type`
-->
<!-- #ICE COMMIT=`t_typeList` -->
<P>
<INPUT TYPE="submit" NAME="Display" VALUE="Display"
    ALT="Show Plays of this type">
</FORM>
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->
</BODY>
</HTML>
```

Extracting Play Types from Plays Table

There are three interesting areas in this example. The first is in the select statement:

```
SQL=`select distinct type from plays`
```

Here we are selecting the various play types from the table (we need to specify distinct because we have denormalized the schema).

Using a Selector Control to Display Distinct Play Types

Next, we see that the TYPE of the result set has been changed from the default (TABLE) to the type SELECTOR. This automatically generates the HTML tags required for a selector control. We use the ATTR keyword to set the NAME variable for this selector control:

```
TYPE=`SELECTOR`
ATTR=`NAME=e_type`
```

Committing the Transaction

Finally, we tidy up the transaction immediately by committing it at once (there is no need to hold it open, in contrast to the one in the previous document):

```
<!-- #ICE COMMIT=`t_typeList` -->
```

Having set the e_type HTML variable on the form, we now include a Submit button to send the successful controls (in particular e_type) to the next (display) document, myplay_subSet.html. We examine this document next.

Displaying a Subset of Table Rows by Selector

The myplay_subSet.html document will be used to display those plays required, as specified by the e_type variable. This is used in the where clause of the select statement:

```
SQL=`select * from plays where
    type = ':e_type'`
```

We also include a hyperlink back to the type selection page:

```
<A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeList.html]">Select a new play
    type</A>
```

In every other respect, this document is identical to the first example we saw. Extending this example to wrap around to the beginning when the end of the result set is reached is left as an exercise to the reader.

Displaying Plays Based on Type

Include the following code in the myplay_subset.html file:

```
<HTML>
<HEAD>
<TITLE>Shakespeare's Plays by Type</TITLE>
</HEAD>
<BODY>
<H1>Shakespeare's Plays by Type</H1>

<FORM ACTION="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_subSet.html]" METHOD="POST">

<!-- #ICE
    DATABASE = `icetutor`
    SQL=`select * from plays where type = ':e_type'`
    TRANSACTION=`Complete`
    CURSOR=`Works`
    ROWS=`5`
    TYPE=`TABLE`
-->

<!-- #ICE DECLARE=`session.e_playTxn=Complete` -->

<!-- #ICE DECLARE=`session.e_playTxnCommitted=FALSE` -->

<P>

<A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeList.html]">Select a new play
    type</A>
<P>

<INPUT TYPE="submit" NAME="More" VALUE="More"
     ALT="Show More Plays of this type">
</FORM>
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->
</BODY>
</HTML>
```

In this example, we have seen how we can communicate values between two (or more) Web Deployment Option HTML documents—presenting a simple refinement of search criteria to the user based on data in the database.

In the next example, we see how to achieve a similar effect, this time using hyperlinks instead of a selector control.

Creating Automatically-Generated Hyperlinks

This example is a simple variation of the previous one, using hyperlinks in place of a selector control. To do this, we replace the SELECTOR keyword with the LINKS keyword. This automatically generates the HTML tags to create a hyperlink for each row and pass a variable to the target page. The name of the variable is that of the selected column and the value is the contents of the column for that row.

In addition, we set a return address for the following page so that it can link back to this one. That way, we can use the same sub-page from more than one page, creating in effect a doubly-linked list albeit with only two members.

Using a Hyperlink to Obtain Play Type

Add the following code to the myplay_typeLink.html page:

```
<HTML>
<HEAD>
<TITLE>Hyper Link to Shakespeare's Plays by
    type</TITLE>
</HEAD>
<BODY>
<H1>Hyper Link to Shakespeare's Plays by type</H1>
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_TxnCndCmt_h.html]` -->
<!-- #ICE
        DATABASE = `icetutor`
        SQL=`select distinct type from plays`
        TRANSACTION=`t_type`
        CURSOR=`c_type`
        ROWS=`10`
        TYPE=`PLAIN`
        LINKS=`type,/ice-bin/oiice.dll/my_playgroup/
            my_plays[myplay_typeLinkSubSet.html]`
-->
<!-- #ICE COMMIT=`t_type` -->
<!-- #ICE DECLARE=
    `session.e_return=myplay_typeLink.html` -->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->
</BODY>
</HTML>
```

Using the LINKS Keyword to Generate Distinct Play Type Hyperlinks

The required tags to generate the hyperlinks (one per row) are generated by the LINKS keyword as follows:

```
LINKS=`type,/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLinkSubSet.html]`
```

The first parameter, type, specifies the column name. This sets the name of the variable that will be passed to the target document. The second argument specifies the document that is the target of the link. The following construct produces hyperlinks given the contents of the plays table. The data in the plays table is shown in Plays Tutorial Application Data.

```
<A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLinkSubSet.html]?type=comedy">
    comedy</A>

<A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLinkSubSet.html]?type=history">
    history</A>

<A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLinkSubSet.html]?type=tragedy">
    tragedy</A>
```

Using a Session Variable to Return to Current Document

We next declare a new session variable, e_return, because the next few examples use the same page to list the required plays and we would like to return to the current document. Therefore, the variable takes on the value myplay_typeLink.html, which is the name of this document:

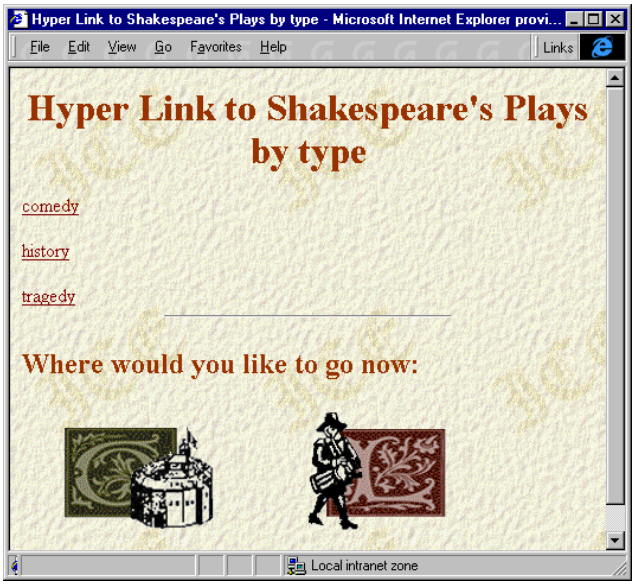
```
<!-- #ICE DECLARE=
    `session.e_return=myplay_typeLink.html` -->
```

We next examine the document that is the target of all three generated links from this document.

Displaying a Subset of Table Rows by Hyperlink

This document is virtually the same as the sub-select document in the selector control example. The variable name in the where clause has been changed to be the HTML variable TYPE and we include a parameterized hyperlink back to the calling page using the Web Deployment Option session variable e_return. That said, it is a trivial exercise to modify the selector control's sub-select document to have the same functionality as this document.

Hyperlink selector controls are shown in the following page:



Displaying Plays Based on Type

Let's now include the following code in the file named myplay_typeLinkSubSet.html:

```
<HTML>
<HEAD>
<TITLE>Link generated sub-set of Shakespeare's Plays
    by type</TITLE>
</HEAD>
<BODY>
<CENTER>
<H1>Link generated sub-set of Shakespeare's Plays by
    type</H1>
</CENTER>

<FORM ACTION="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLinkSubSet.html]" METHOD="GET">
<!-- #ICE DECLARE=`session.e_playTxn=Complete` -->
<!-- #ICE DECLARE=`session.e_playTxnCommitted=FALSE`
-->
<!-- #ICE REPEAT
        DATABASE = `icetutor`
        SQL=`select * from plays where type = ':type'`
        TRANSACTION=`Complete`
        CURSOR=`Works`
        ROWS=`5`
        TYPE=`TABLE`
-->
<P>
<!-- #ICE VAR=`
    <A HREF="/ice-bin/oiice.dll/my_playgroup/
        my_plays[:e_return]">Select a new play
        type</A>`
-->
<P>
<INPUT TYPE="submit" NAME="More" VALUE="More"
    ALT="Show more plays">
<!-- #ICE VAR=`<INPUT TYPE="hidden" NAME="type"
    VALUE=":type">` -->
</FORM>

<!-- #ICE REPEAT
    INCLUDE=`my_plays[myplay_SessionControl_h.html]`
-->
</BODY>
</HTML>
```

Using a Dynamic Hyperlink to Return to Calling Page

Expanding the Web Deployment Option session variable e_return creates the hyperlink that returns the user to the calling page. The VAR keyword takes any text (including HTML text) containing Web Deployment Option variables and replaces them with their contents. Here we make use of this to generate a hyperlink dynamically:

```
<!-- #ICE VAR=` <A HREF="/ice-bin/oiice.dll/
    my_playgroup/my_plays[:e_return]">Select a new
    play type</A>` -->
```

With this pair of documents, we have ended up with text links based on the contents of the table making the page dynamic. As an improvement, we would like to have an image to click rather than plain text. This is demonstrated in the next section.

Creating Graphical Hyperlinks

Our first attempt at building a hyperlink list of play types was simple and effective, but we would much rather use a graphical link to do this. In this example, we build a link by embedding the HTML tags directly in the select statement.

The document using graphical hyperlinks is shown below:

Using Graphical Hyperlinks to Obtain Play Type

We will now set the return address as before, in the myplay_typeGLink.html file. Enter the following code:

```
<HTML>
<HEAD>
<TITLE>Graphical Hyper Link to Shakespeare's Plays by
    type</TITLE>
</HEAD>
<BODY>
<H1>Graphical Hyper Link to Shakespeare's Plays by
    type</H1>

<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_TxnCndCmt_h.html]` -->
```

```
<!-- #ICE REPEAT
        DATABASE = `icetutor`
        SQL=`select distinct
            '<A HREF="/ice-bin/oiice.dll/my_playgroup
            /my_plays[myplay_typeLinkSubSet.html]?
            type=', type, '"><IMG SRC="/
            ice-bin/oiice.dll/my_playgroup/
            my_plays[', type,'.gif]" alt="',
            type,'"></A>' from plays`
        TRANSACTION=`t_type`
        CURSOR=`c_type`
        ROWS=`-1`
        TYPE=`PLAIN`
-->
<!-- #ICE COMMIT=`t_type` -->
<!-- #ICE DECLARE=
    `session.e_return=myplay_typeGLink.html` -->

<!-- #ICE REPEAT
    INCLUDE=`my_plays[myplay_SessionControl_h.html]` -->
</BODY>
</HTML>
```

Embedding Static Tags for Hyperlinks in the SQL Statement

The first thing we notice is that the select statement has become far more complicated than it was. We are making use of the ability to embed static text within the statement to embed the tags for a hyperlink. We select the column named "type" three times in all. First, to pass in as the value of the variable passed to the target document; second, as the name of the image file (we hard code the extension); and finally, as the alternative text to the image as follows:

```
SQL=`select distinct
    '<A HREF="/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLinkSubSet.html]?type=', type,
     '">
    <IMG SRC="/ice-bin/oiice.dll/my_playgroup/
        my_plays[', type, '.gif]" alt="', type, '">
    </A>'
    from plays`
```

The first line introduces the select. The hyperlink anchor tag with the variable type passes the contents of the "type" column as the value. The image tag <IMG SRC> uses the contents of the type column in two places—the first as the name of the image file (with the hard-coded extension) and the second as the alternate text for the image.

Both the link and the value of the HTML variable type are built dynamically in this document. We note that, although in this case the image files are constrained to have the same names as the various play types, this is:

- Not necessarily a bad thing
- Easily changed by joining with an image table

We set the value of the e_return session variable so as to reuse the display document from before.

This document achieved the aim of dynamically creating a set of graphical hyperlinks, but at the expense of placing HTML code within the select statement. We would prefer to abstract the HTML away from the SQL. The keywords, HTML and SWITCH, make this possible.

Creating Switch Image Links

Building up the hyperlink in the SQL select statement can be confusing and make code very difficult to maintain. We would rather separate the HTML code from the SQL statement and this is just what the HTML keyword allows us to do. For each record returned, we execute a SWITCH statement, embedding the image file name (now independent from the value of the type column) and, in addition, different alternative text for each image.

Using the HTML and SWITCH Keywords to Generate Distinct Play Type Graphical Hyperlinks

In the final example, include the following code in the myplay_typeGSLink.html file:

```
 <HTML>
 <HEAD>

<TITLE>Graphical Switched Hyper Link to Shakespeare's
    Plays by type</TITLE>
</HEAD>
<BODY>
<H1>Graphical Switched Hyper Link to Shakespeare's
    Plays by type</H1>

<!-- #ICE REPEAT
    INCLUDE=`my_plays[myplay_TxnCndCmt_h.html]` -->

<!-- #ICE REPEAT
    DATABASE = `icetutor`
    SQL=`select distinct type from plays`
    TRANSACTION=`t_type`
    CURSOR=`c_type`
    ROWS=`10`
    TYPE=`UNFORMATTED`
    HTML=`<P><!-- #ICE SWITCH=``:type``
        CASE ``comedy``=``<A HREF=
            /ice-bin/oiice.dll/my_playgroup
            /my_plays[myplay_typeLinkSubSet.html]?
            type=:type><IMG SRC="/
            ice-bin/oiice.dll/my_playgroup/
            my_plays[comedy.gif]" alt="Laugh at the
            Comedy plays"></A>``
        CASE ``tragedy``=``<A HREF=
            /ice-bin/oiice.dll/my_playgroup
            /my_plays[myplay_typeLinkSubSet.html]?
            type=:type><IMG SRC="/ice-bin/oiice.dll
            /my_playgroup/my_plays[tragedy.gif]" alt=
            "Be moved by the tragedy plays"></A>``
        CASE ``history``=``<A HREF=
            /ice-bin/oiice.dll/my_playgroup
            /my_plays[myplay_typeLinkSubSet.html]?
            type=:type><img src="/ice-bin/oiice.dll
            /my_playgroup/my_plays[history.gif]" alt=
            "Look back at the history plays"></A>``
        -->`
-->
```

```
<P>
<!-- #ICE COMMIT=`t_type` -->
<!-- #ICE DECLARE=`session.e_return=play_typeGSLink.html` -->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->
</BODY>
</HTML>
```

The first thing we notice is that the select statement has gone back to being simple once more. Secondly, there is a large block of HTML code appearing at the end of the entire statement, following the HTML keyword. This implements a switch taking the value of the type column as its variable. There are three recognized play types in this application: "comedy," "tragedy," and "history." There is a CASE for each.

Note that we have not made use of the DEFAULT case in this example. For each case, we specify the same target document, pass the variable type in, and set individual picture files and alternative text for them. The SWITCH statement appears within the HTML statement and this requires us to:

- Use the REPEAT keyword in the SQL statement
- Double-up the grave quotes (`)

Doubling Grave Quotes

The REPEAT informs Web Deployment Option that it must evaluate a sub-statement (SWITCH in this case) and because of this, we must protect the grave quotes in the SWITCH statement by doubling them up, as follows:

```
CASE ``comedy``=
``<A HREF=/ice-bin/oiice.dll/my_playgroup/
    my_plays[myplay_typeLinkSubSet.html]?type=:type>/
    <IMG SRC="/ice-bin/oiice.dll/my_playgroup
    my_plays[comedy.gif]" alt="Laugh at the Comedy
    plays">
    </A>``
```

Each CASE of the SWITCH statement is similar—here we examine the comedy case. First, we build a hyperlink to the display document, myplay_typeLinkSubSet.html, and pass in the variable type with the value ":type," which comes from the column of that name. We then provide an image for this link and this is now no longer constrained to be the value of the column; neither is the alternative text.

We have now covered most of the Web Deployment Option macro keywords and features. You can see how you can use them to develop applications that are both robust and visually appealing. In the next section, we will build another very popular type of application—an Internet shopping application.

Designing an Internet Shopping Application

In this section, we will examine the code in the Plays application provided with Web Deployment Option for shopping for Shakespeare products at the Globe Boutique using the Internet.

The Globe Boutique Home Page

The home page for the Globe Boutique has two main jobs. It must list all the items for sale, which are listed in the table named play_item. This is a standard type of select statement that we have seen before.

In addition, the home page must allocate an order number for the user (if one does not already exist) and this is achieved by invoking a user-defined function extension to the ICE server (myplay_neworder.sc file). The function invokes a database procedure (in the myplay_newOrder.sql file) which increments a count in a table. We will examine each of these files, starting with the procedure. For clarity, error-checking code has been left out.

Windows

Note: This example assumes that the function is to be built for the Windows platform. Check to see if Ingres embedded SQL/C supports your C compiler version.

The my_new_order procedure produces a new order number by updating the order number in the counters table. This should be executed in a transaction on its own and the order number later used as needed in another transaction. This is exactly what happens when the Web Deployment Option extension, My_NewOrder(), executes it.

We also include a general shop action page, which activates those links set to "Yes" in the parameter list.

Creating the Tables for the Globe Boutique Application

In the Plays application, the play_items, plays_order, and play_counters are used in the Globe Boutique shopping segment. For consistency, you can recreate the tables for the tutorial and rename them with a "my" prefix (that is, myplay_items, etc.). You could also simply use the Plays tables and refer to these in your code.

To begin with, we write the database procedure to generate monotonically increasing order numbers.

Creating the New Order Procedure

You will now create an SQL script that will create the my_new_order procedure. Beneath the myplays directory, create a subdirectory named src. Then, create a file named myplay_newOrder.sql file and enter the following statements:

```
/* Procedure my_new_order */
create procedure my_new_order as
declare
next integer not null;
begin
    select value into :next from counters
        where name = 'order';
next = next + 1 ;
update counters set value = :next
    where name = 'order';
return :next;end;
```

The my_new_order database procedure is invoked on its own in a transaction by the My_NewOrder() server extension, which returns the value as a string in the out_OrderNumber variable.

Creating the New Order Extension Header File

The file myplay_NewOrder.sc includes the header file, myplay_NewOrder.h, to define return types and the ice_function_table function description table, reproduced here:

```
/*
** Name: myplay_NewOrder.h
**
** Description:  Defines the types used for the
** extension server      functions
*/
# include <windows.h>

# define ICE_EXT_API __declspec(dllexport)

typedef char*       ICE_STATUS;
typedef ICE_STATUS  (*PFNEXTENSION) (char**, BOOL*,
    char **);

typedef struct ice_function_table
{
    char*       pszName;
    char**      pszParams;
}SERVER_DLL_METHOD, *PSERVER_DLL_METHOD;

typedef ICE_STATUS
    (*PFNINITIALIZE)(PSERVER_DLL_METHOD*);
```

Creating the New Order Extension

The ICE server defines an interface to which all extension functions must be written. This includes a defined entry point that returns a structure describing the function and its parameters. This structure and the function are to be reproduced in the file myplay_NewOrder.sc:

```
/**
** Name: myplay_NewOrderNr.sc
**
# include "play_NewOrder.h"
*/
# defineMAX_SIZE        20
/**
** Parameter name list.
** A NULL pointer terminates the list.
*/
static char* pszNewOrderParams[] =
    {"out_orderNumber", NULL};

/**
 ** Function Description
*/
static SERVER_DLL_METHOD FunctionTable[] =
{
        { "newOrder", { pszNewOrderParams } },
        { NULL }
};

/*
 * Name: InitICEServerExtension
 *
 * Description:
 *      Mandatory function for providing method
 *      description to the server.
 *
 * Inputs:
 *      None.
 *
 * Outputs:
 *      ppServerDllMethod: pointer to the function
 *      description structure.
 *
 * Returns:
 *      pointer to error text
 *      NULL on success
 */
ICE_EXT_API ICE_STATUS
InitICEServerExtension(PSERVER_DLL_METHOD* ppServerDllMethod)
{
    ICE_STATUS status= NULL;
    *ppServerDllMethod = FunctionTable;
    return status;
}
```

```
/*
 * Name: newOrder
 *
 * Description:
 *      Return the next order number
 *
 * Inputs:
 *      None.
 *
 * Outputs:
 *      out_OrderNumber
 *
 * Return:
 *      pointer to error text
 *      NULL on success
 */
ICE_EXT_API ICE_STATUS
newOrder(char** out_OrderNumber, BOOL* print, char**
    context)
{
    ICE_STATUS  status = NULL;

    *print = FALSE;
    /*
    ** if first invocation allocate some memory for
    ** the result
    */
    if (*context == NULL)
    {
        exec sql begin declare section;
        long x;
        exec sql end declare section;

        *context =
            HeapAlloc(GetProcessHeap(),
            HEAP_ZERO_MEMORY, MAX_SIZE);
        if (*context == NULL)
        {
            return ("Memory error\n");
        }

    exec sql connect 'icetutor' identified by
        'icedbuser';
    exec sql execute procedure my_new_order into :x;

    exec sql commit;
    exec sql disconnect;

    sprintf (*context, "%d", x);

    *out_OrderNumber = *context;
    *print = TRUE;
    }
else
{
    HeapFree (GetProcessHeap(), 0, *context);
    *context = NULL;
}

return (status);
}
```

Entry Point

There is some "housekeeping" that needs to be done when writing an ICE server extension function. There must be an InitICEServerExtension() function, its purpose being to return a structure describing the function. This structure contains the name of the function and its parameter list. In our case, the function is My_NewOrder() and it exports one value parameter, out_OrderNumber, and a context pointer to the memory location used, context.

Generating a New Order Number

When My_NewOrder() is first invoked, the context pointer is set to NULL, indicating that it the must allocate some memory to hold the value to be returned, it returns a pointer to that memory in the context variable. The next order number is returned in the out_OrderNumber parameter.

Clean-Up

My_NewOrder() frees context memory when it is invoked with a non-NULL value for context.

Building the New Order Extension

Windows

The extension function must be built as a dynamic link library (for the Windows platform) and installed in the files\dynamic directory in the Ingres system area (addressed by the environment variable II_SYSTEM).

Make File Used to Build DLLs

The file named makefile (in the \ingres\ice\plays\src directory) is used to build the DLL. Its contents are shown below:

```
CFLAGS=-c $(CDEBUG) -MD -D_X86_=1 -DWINVER=0x0400 -DWIN32 -D_WIN32
LFLAGS=/DLL $(LDEBUG)

all: myplay_NewOrder.dll

myplay_NewOrder.dll: myplay_NewOrder.obj
    link $(LFLAGS) /out:$@ $** ws2_32.lib \
    $(II_SYSTEM)\ingres\lib\libingres.lib

myplay_NewOrder.obj: myplay_NewOrder.c
    myplay_NewOrder.h
    cl $(CFLAGS) $*.c

myplay_NewOrder.c: myplay_NewOrder.sc
    esqlc $*.sc

install: myplay_NewOrder.dll
    copy myplay_NewOrder.dll
    "$(II_SYSTEM)\ingres\files\dynamic\
    myplay_NewOrder.dll"
```

The "install" target copies the library to the appropriate place, once it has been successfully built (target "all").

Using the Extension Function on the Web Page

Once the dynamic link library has been successfully built and installed, it can be used in a Web document. We chose to issue every visitor to the Globe Shop home page a unique order number. In addition, we must ensure that an existing visitor who has confirmed an order is issued a *new* order number, in case they wish to come back and order more items.

Globe Boutique Home Page

Enter the following code into the myplay_shopHome.html file:

```
<HTML>
<HEAD>
<TITLE>Globe Boutique</TITLE>
</HEAD>
<BODY>
<H1>Globe Boutique</H1>
The Globe Boutique is where you can purchase all your
    favorite Globe memorabilia.
<P>
Visit often to find that gift for the person in your
    life who always seems to have everything
<P>
Genuine quality products and gifts with a unique
    cultural heritage
<P>
<H2>Instructions</H2>
Select an item from the list below to view its
    description.

<!-- #ICE REPEAT IF (DEFINED (e_shopTxn))
    THEN=``
    ELSE=`<!-- #ICE DECLARE=
        ``session.e_shopTxn=NOT-OPEN`` -->`
-->

<!-- #ICE REPEAT IF (DEFINED (e_orderNumber) AND
    `:e_shopTxn` != `COMPLETE`)
    THEN=``
    ELSE=`<!-- #ICE REPEAT
        FUNCTION=``play_NewOrder.newOrder``
        HTML=``<!-- #ICE
        DECLARE=````session.e_orderNumber
            =:out_orderNumber````
        -->``
    -->`
-->
<!-- //enable to see order nr ICE VAR=`<P>DEBUG:
    Order nr is :e_orderNumber` -->
<!-- #ICE
        DATABASE = `icetutor`
        SQL=`set lockmode session where readlock =
            nolock;
            select id, name from play_item`
        TRANSACTION=`Shoppe`
        CURSOR=`Keeper`
        ROWS=`-1`
        TYPE=`TABLE`
```

```
        HEADERS=`id,Reference,name,Article`
        LINKS=`id,my_plays[myplay_shopDescribe.html]`
-->
<!-- #ICE COMMIT=`Shoppe` -->
<!-- Standard Shop Actions -->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_shopAction_h.html]?View=Yes` -->
<!-- Standard Session Control -->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->
</BODY>
</HTML>
```

Checking Shop Entry Variable

We first check to see if the shop has been entered for the first time by checking for the existence of the e_shopTxn variable:

```
<!-- #ICE REPEAT IF (DEFINED (e_shopTxn))
    THEN=``
    ELSE=`<!-- #ICE DECLARE=
        ``session.e_shopTxn=NOT-OPEN`` -->`
-->
```

Checking Order Number and Transaction

If the variable does not exist, it is created and given the value "NOT-OPEN" to indicate that a (shopping) transaction has yet to be opened.

We next check to see if an order number, e_orderNumber, exists and if the transaction has been completed.:

```
<!-- #ICE REPEAT IF (DEFINED (e_orderNumber) AND
    `:e_shopTxn` != `COMPLETE`)
```

Issuing a New Order Number

If no order number exists or the transaction is not complete, we need to issue a new order number using the function My_NewOrder(), to be found in the My_Play_NewOrder library.

Note: Do not add the .dll or .so extension to the library name:

```
<!-- #ICE REPEAT FUNCTION=``my_play_NewOrder.my_newOrder``
```

Since the function invocation is part of another Web Deployment Option macro statement, we must double up the grave quotes.

We then set a session variable to hold the order number and terminate the statement:

```
HTML=``<!-- #ICE
    DECLARE=````session.e_orderNumber=:out_orderNumber    ````
        -->``
    -->`
-->
```

When we declare the e_orderNumber variable, we are nested two levels down, and so need to double-up the grave quotes again. The number of grave quotes you need is easily calculated as 2 raised to the power '*level*', where *level* is the nesting level, here 2 because we start with level = 0 (2 raised to the power 0 is 1).

Creating Links to Item Descriptions

Having allocated the new order number, we perform a select with links to create links on the item identifier to a page providing more detailed description of the item selected. The user can then decide to buy or not to buy.

Providing Shopping Options

At any time in the Globe Shop application, the user can move to any of the other pages. This is accomplished by including the myplay_shopAction_h.html file, which takes the following parameters:

Parameter	Meaning
View	View contents of shopping bag
Confirm	Confirm order
Remove	Empty the shopping bag

The myplay_shopAction_h.html file is included as follows:

```
<!-- Standard Shop Actions: activate View bag only
-->

<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_shopAction_h.html]?View=Yes` -->
```

Activating Option to View Bag Contents

Here we only activate the View Bag Contents option; the Return to Products option to return to the product list is always valid. The action file, play_shopAction_h.html, is shown below. Add this code to the myplay_shopAction_h.html file:

```
<!-- shopAction_h.html:
    Activate links as appropriate. Link is active if
    the variable of that name is set to Yes, else
    inactive
-->
<HR WIDTH="50%">
<TABLE BORDER=0 CELLSPACING=4>
<TR>
<TD>
<!-- #ICE IF (`:View` == `Yes`)
    THEN=`<A HREF="my_plays[myplay_shopView.html]">
        View Bag Contents</A>`
    ELSE=`View Bag Contents`
-->
<TD>
<A HREF="my_plays[myplay_shopHome.html]">Return to
    Products</A>
<TD>
<!-- #ICE IF (`:Confirm` == `Yes`)
    THEN=`<A HREF="my_plays[myplay_shopConfirm.html]">
        Place Order</A>`
    ELSE=`Place Order`
-->
<TD>
<!-- #ICE IF (`:Remove` == `Yes`)
    THEN=`<A HREF="my_plays[myplay_shopRemove.html]">
    Empty Bag</A>`
    ELSE=`Empty Bag`
```

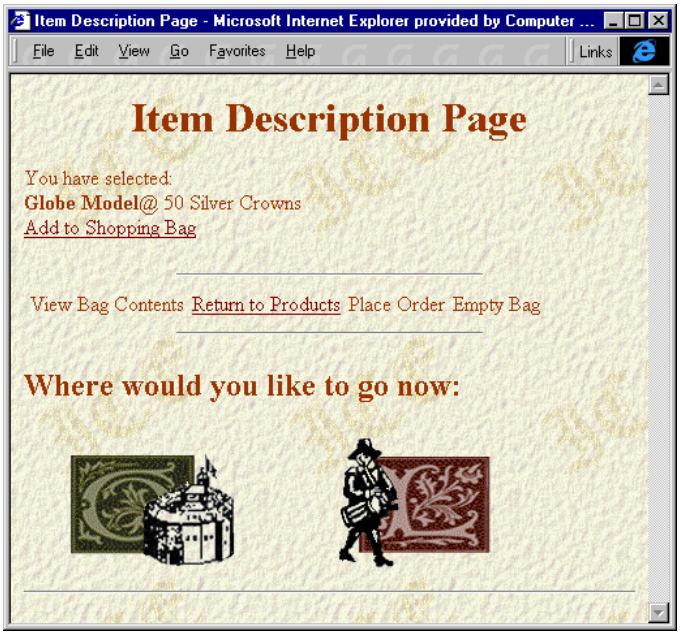
```
-->
</TABLE>
```

This file builds a table and sets the contents of each cell to be either an active link or an inactive link (normal text) depending on the values of the variables passed in. We will examine the item description page next.

Displaying an Item Description

The description page selects the required product, passed in as the HTML variable ID, and displays more information about the item—in this case, the price. (An application typically presents more information about a product, such as textual description and photo. In the Plays application, the price is used as a sample field.)

The Item Description Page appears as follows:



In addition, a link is provided to add the item to the shopping bag and another to return to the Globe Shop home page. The add item link passes the item ID on to the confirmation page.

Displaying a Selected Item's Description

The following code appears in the play_shopDescribe.html file. Proceed by adding this code to the myplay_shopDescribe.html file.

```
<HTML>
<HEAD>
<TITLE>Item Description Page</TITLE>
</HEAD>
<BODY>
<CENTER>
<H1>Item Description Page</H1>
</CENTER>

<!-- #ICE REPEAT
        DATABASE = `icetutor`
        SQL=`select id, name, cost from play_item
            where id = :id`
        TYPE=`UNFORMATTED`
        HTML=`<p>You have selected:
            <BR><B>:name</B>@:cost Silver Crown
            <!-- #ICE IF ( ``1`` !=``:cost``)
            THEN=``s`` -->
                <BR><A HREF=
                "my_plays[myplay_shopAdd.html]?
                id=:id">Add to Shopping Bag</A>`
-->
<P>

<!-- Standard Shop Action -->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_shopAction_h.html]` -->
<!-- Standard Session Control -->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->
</BODY>
</HTML>
```

First of all, we select the ID, name, and cost of the product from the play_item table. The ID is specified in the where clause, passed in with the HTML variable ID, and the UNFORMATTED type is specified:

```
SQL=`select id, name, cost from play_item
    where id = :id`
TYPE=`UNFORMATTED`
```

We need an unformatted output from the select statement because we want to add the HTML code to add this item to the shopping bag, which we do as follows:

```
HTML=`<P>You have selected:<BR>
    <B>:name</B>
    @ :cost Silver Crown<!-- #ICE IF ( ``1`` !=
        ``:cost``)
        THEN=``s`` -->
    <BR>
    <A HREF=
        "my_plays[myplay_shopAdd.html]?id=:id">
        Add to Shopping Bag</A>`
```

Adding Items to Shopping Bag

The anchor, in the last line, is built up by passing the item identifier held in the HTML variable ID to the next document, myplay_shopAdd.html, with the caption "Add to Shopping Bag".

For that final touch, we test to see if the cost of the item is 1 Silver Crown. If it is not, we pluralize the word Crown, thus resulting in "Crown" or "Crown**s,**" as required.

The following page examines the code necessary to add the item to the shopping bag.

Adding an Item to the Shopping Bag

There are three main sections in the document that adds an item to the shopping bag. The first inserts the required item into the myplay_order order table, along with the user's login ID and the order number. A status of 1 is included.

The meaning of the status column is as follows:

Status	Meaning
1	Item placed in bag
2	Item ordered
3	Order passed to warehouse
4	Order in dispatch
5	Courier confirms delivery

This exercise set uses the first two status values. If you would like to add code to handle the remaining statuses, see Further Exercise in this chapter.

The second section confirms that the item has been added and in the final section, the user can return to the shop (Return to Products) or view the contents of their shopping bag (View Bag Contents).

Order Processing

Add the following code to the myplay_shopAdd.html file:

```
<HTML>
<HEAD>
<TITLE>Add to Bag: Confirmation</TITLE>
</HEAD>
<BODY>
<H1>Add to Bag: Confirmation</H1>
```

```
<!-- #ICE
    DATABASE=`icetutor`
    SQL=`insert into play_order
        (order_nr, user_id, product_id, status)
        values (:e_orderNumber, ':ii_userid', :id, 1)`
        TRANSACTION=`t_shopAdd`
-->
<!-- Flag the transaction as open; we have now added something to the bag -->
<!-- #ICE DECLARE=`session.e_shopTxn=OPEN` -->
<TABLE BORDER=0>
<TR>
<TD BGCOLOR="lime">
The item,
<!-- #ICE REPEAT
    DATABASE=`icetutor`
    SQL=`
        set lockmode session where readlock = nolock;
        select name, cost from play_item where id =
            (select product_id from play_order
            where order_nr = :e_orderNumber
            and user_id = ':ii_userid'
            and product_id = :id)`
    TRANSACTION=`t_shopAddConfirm`
    TYPE=`UNFORMATTED`
    HTML=`<B>:name</B> @<I>:cost Silver Crown
        <!-- #ICE IF ( ``1`` != ``:cost``)
        THEN=``s`` -->
    </I>`
-->

<!-- #ICE COMMIT=`t_shopAddConfirm` -->
    has been added to your shopping bag.
</TD></TR></TABLE>

<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_shopAction_h.html]?View=Yes` -->
<!-- Standard Session Control -->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->

</BODY>
</HTML>
```

Adding Order to myplay_order Table

The first thing this document must do is actually to add the ordered item to the myplay_order table. This is accomplished with an SQL insert statement:

```
SQL=`insert into myplay_order
    (order_nr, user_id, product_id, status)
    values (:e_orderNumber, ':ii_userid', :id, 1)`
TRANSACTION=`t_shopAdd`
```

Opening a Transaction

We do not commit this transaction yet because the user may want to add other items to the order. Furthermore, the user may log out or time out before completing their purchases, at which point the ICE server automatically rolls back any open transactions on our behalf. What we do instead is to set the e_shopTxn session variable to OPEN with the statement shown below:

```
SQL=`insert into myplay_order
    <!-- #ICE DECLARE=`session.e_shopTxn=OPEN` -->
```

Although this is a dummy value, it is meaningful. Remember that when its value is COMPLETE, the shop home page will generate a new order number. We defer committing the transaction until later (on the order confirmation page).

Displaying Ordered Items

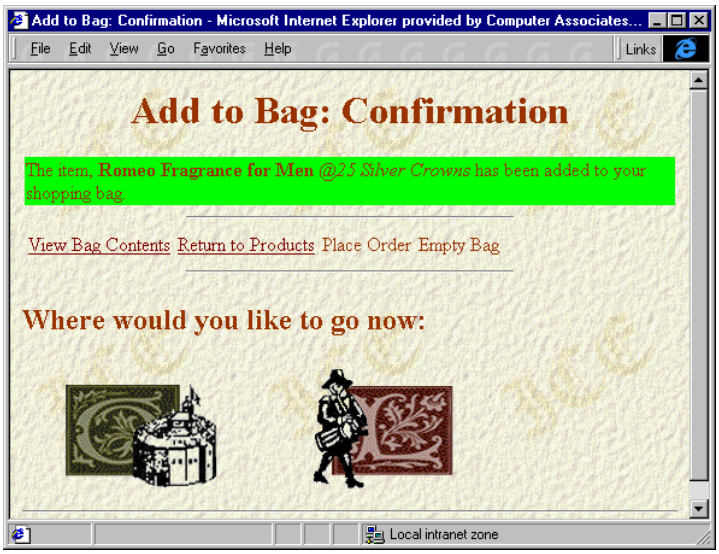
Next, we select the same record from myplay_order and display it in a table with a green background. We place the select statement inside a table cell (<td>). Note that we use readlock = nolock to avoid any locks that may be taken on the table. The transaction, t_shopAddConfirm, is committed immediately—in fact, within the HTML table element.

Here is the table element in its entirety:

```
<TABLE BORDER=0>
<TR>
<TD BGCOLOR="lime">
The item,
<!-- #ICE REPEAT
    DATABASE=`icetutor`
    SQL=`set lockmode session where readlock = nolock;
        select name, cost
        from play_item
        where id =
            (select product_id from play_order
            where order_nr = :e_orderNumber
            and user_id = ':ii_userid'
            and product_id = :id)`
        TRANSACTION=`t_shopAddConfirm`
        TYPE=`UNFORMATTED`
        HTML=`<B>:name</B> @<I>:cost Silver Crown
            <!-- #ICE   IF ( ``1`` != ``:cost``)
                THEN=``s`` -->
        </I>`
-->
<!-- #ICE COMMIT=`t_shopAddConfirm` -->
    has been added to your shopping bag.
</TABLE>
```

The same technique from the previous example has been used to add an "s" to the Silver Crown if the cost is other than one Silver Crown.

The Add to Bag:Confirmation page appears as follows:



Enabling Shop Action Links

In the third section, we enable the View option for the standard set of shop action links:

```
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_shopAction_h.html]?View=Yes` -->
```

Further Exercise

Using the status values 3–5, extend the Web site so that the user, specifying their order number, can track the order through the system. *Hint: At each stage throughout increment the status; orders of status 5 can be purged from the current order table.*

We have already examined the Globe Shop home page, we now move on to the View Bag Contents page.

Displaying Shopping Bag Contents

The user will most likely want to view the contents of their shopping bag before confirming or canceling the order. The View Bag Contents page provides that all important functionality and consists of a select to present the entire contents of the user's bag, it also totals up the cost and activates the Remove and Confirm options from the standard shop actions.

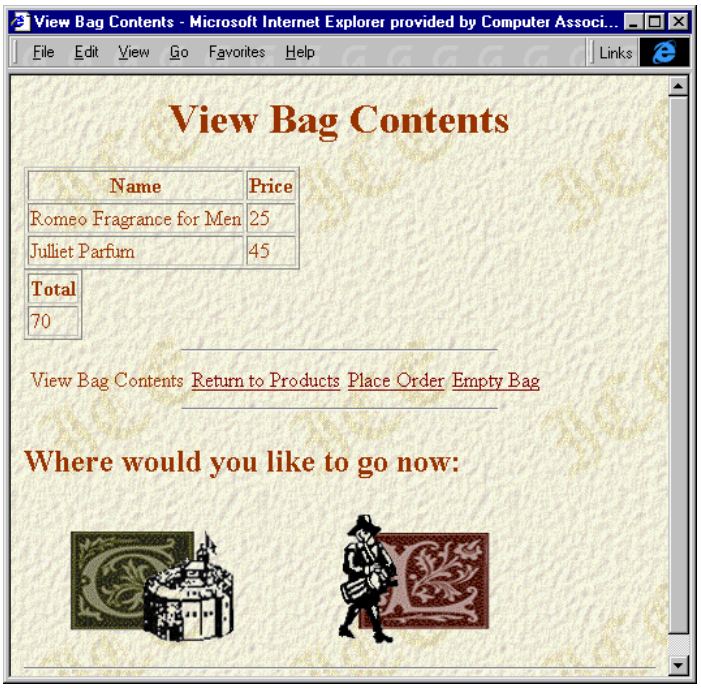
Note that the Empty Bag and Remove options—which are displayed but inactive on the page—can be activated by setting the value of the variables Remove and Confirm, respectively, to "Yes".

Viewing Bag Contents

Enter the following code into the myplay_shopView.html file:

```
<HTML>
<HEAD>
<TITLE>View Bag Contents</TITLE>
</HEAD>
<BODY>
<H1>View Bag Contents</H1>

<!-- #ICE
    DATABASE=`icetutor`
    SQL=`set lockmode session where readlock = nolock;
        select name, cost
        from play_item, play_order
        where id = product_id and
            (order_nr = :e_orderNumber and
            user_id =   ':ii_userid')`
    HEADERS=`name, Name, cost, Price`
    TYPE=`TABLE`
-->

<!-- #ICE
    DATABASE=`icetutor`
    SQL=`set lockmode session where readlock = nolock;
        select sum(cost)
        from play_item, play_order
        where id = product_id and
            (order_nr = :e_orderNumber and
            user_id = ':ii_userid')`
    HEADERS=`col1, Total`
    TYPE=`TABLE`
-->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_shopAction_h.html]?
    Remove=Yes&Confirm=Yes` -->
<!-- Standard Session Control -->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->
</BODY>
</HTML>
```

The View Bag Contents page appears as follows:



Naming Column Headings

When Web Deployment Option does not have a column name specified in the select statement (for example, when we use a function), it names the columns col1, col2, and so on. In this case, we sum the cost column and rather than having the total cost of our order shown with the heading col1, we use the HEADERS keyword to replace the generated column name with a more descriptive one, "Total." The code fragments appear below:

```
select sum(cost)
...
HEADERS=`col1, Total`
```

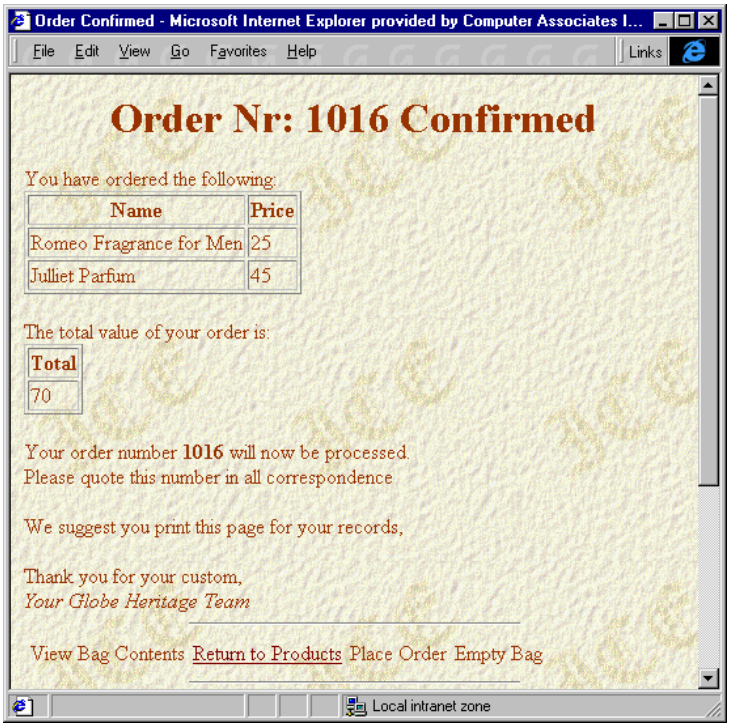
This page requires that we pass two variables into the parameterized included file myplay_shopAction_h.html; the variables must be separated by the & symbol:

```
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_shopAction_h.html]?Remove=
    Yes&Confirm=Yes` -->
```

Once satisfied with the contents of the shopping bag, the user will next want to confirm their purchases.

Confirming an Order

When the user confirms their purchases, this must be reflected in the database by updating the status of the order. The transaction as a whole must then be committed and signaled to the home page so that if the user returns to the home page, they can start again with a new order number. Then we display the contents of the order, the cost, and the order number, for reference:



Committing the Transaction

To accomplish this, enter the following code into the myplay_shopConfirm.html file:

```
<HTML>
<HEAD>
<TITLE>Order Confirmed</TITLE>
</HEAD>
<BODY>
<H1>Order Nr: <!-- #ICE VAR=`:e_orderNumber` -->
Confirmed</H1>

<!-- #ICE
    DATABASE=`icetutor`
    SQL=`update play_order
        set status = 2
        where order_nr = :e_orderNumber
        and user_id = ':ii_userid'`
    TRANSACTION=`t_shopAdd`
-->
<!-- Commit the transaction -->
<!-- #ICE COMMIT=`t_shopAdd` -->
<!-- Set the transaction to 'complete' as it is now committed -->
<!-- #ICE DECLARE=`session.e_shopTxn=COMPLETE` -->
```

```
<!-- Show the products ordered: -->
<P>
You have ordered the following:
<BR>
<!-- #ICE
    DATABASE=`icetutor`
    SQL=`set lockmode session where readlock = nolock;
        select name, cost
        from play_item, play_order
        where id = product_id and
            (order_nr = :e_orderNumber and user_id =
            ':ii_userid')`
    HEADERS=`name, Name, cost, Price`
    TYPE=`TABLE`
-->

<!-- Show the total cost: -->
<P>
The total value of your order is:
<BR>
<!-- #ICE
    DATABASE=`icetutor`
    SQL=`set lockmode session where readlock = nolock;
        select sum(cost)
        from play_item, play_order
        where id = product_id and
            (order_nr = :e_orderNumber
            and user_id = ':ii_userid')`
    TRANSACTION=`t_shopConfirmList`
    HEADERS=`col1, Total`
    TYPE=`TABLE`
-->
<!-- Commit the transaction -->
<!-- #ICE COMMIT=`t_shopConfirmList` -->
<P>
<!-- #ICE VAR=`Your order number <B>:e_orderNumber</B> will now be processed.<BR>
Please quote this number in all correspondence` -->
<P>
We suggest you print this page for your records,
<P>
Thank you for your custom,
<BR>
<I>Your Globe Heritage Team</I>
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_shopAction_h.html]` -->
<!-- Standard Session Control -->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_SessionControl_h.html]` -->
</BODY>
</HTML>
```

Completing the Transaction

Once the status of the order has been updated in the myplay_order table, we commit the transaction and then update the value of the e_shopTxn session variable to "COMPLETE."

```
<!-- #ICE DECLARE=`session.e_shopTxn=COMPLETE` -->
```

This will cause the code in the shop home page to issue a new order number should the user return there.

If the user decides *not* to continue with their order, they will proceed to empty the contents of their bag.

Rolling Back a Transaction

Canceling an Order

If the user decides not to buy the products that have been placed in the bag, they follow the link to this page, where the transaction is rolled back. The transaction is not flagged as being completed though, because the user may simply want to start again. If this is the case, we do not need to generate a new order number. We can continue with the one already issued.

As a confirmation that the order has been rolled back, we display the empty table and the total cost of the order (this is not something that one would do in reality, it is present as an artifact of our example).

The following code should be entered into the file named myplay_ShopRemove.html:

```
<HTML>
<HEAD>
<TITLE>Empty Bag</TITLE>
</HEAD>
<BODY>
<H1>Empty Bag</H1>
<!-- #ICE ROLLBACK=`t_shopAdd` -->

<P>Your shopping bag now contains:
<!-- #ICE
    DATABASE=`icetutor`
    SQL=`set lockmode session where readlock = nolock;
        select name, cost
        from play_item
        where id =
            (select product_id from play_order
                where order_nr = :e_orderNumber
                and user_id = ':ii_userid')`
    TYPE=`TABLE`
-->

<!-- #ICE
    DATABASE=`icetutor`
    SQL=`set lockmode session where readlock = nolock;
        select sum(cost) as Total
        from play_item
        where id =
            (select product_id from play_order
            where order_nr = :e_orderNumber
            and user_id = ':ii_userid')`
    TYPE=`TABLE`
-->
<!-- Standard Shop Action -->
<!-- #ICE REPEAT INCLUDE=
    `my_plays[myplay_shopAction_h.html]` -->
<!-- Standard Session Control -->
<!-- #ICE REPEAT        INCLUDE=`my_plays[myplay_SessionControl_h.html]` -
    ->
</BODY>
</HTML>
```

Rolling Back the Transaction

This page demonstrates the use of the ROLLBACK keyword and this appears as the first active element in the file:

```
<!-- #ICE ROLLBACK=`t_shopAdd` -->
```

The other features have already been examined. We note that the e_shopTxn session variable has not been updated. This means that should the user wish to start again, they will not be issued with a new order number by the shop home page. Since they have just rolled back their transaction, the order number is effectively unused and still valid.

This completes the electronic commerce example and our discussion of the Globe web site.

Plays Tutorial Application Data

The plays table, used with the sample Plays tutorial application, resides in the icetutor database. The following table shows the rows in the plays table:

Comporder	Title	Playright	Performed	Acts	Type
1	The Two Gentlemen of Verona	Shakespeare	1598	5	comedy
2	The Taming of the Shrew	Shakespeare	(null)	5	comedy
3	Henry VI part 1	Shakespeare	1591	5	history
4	Henry VI part 3	Shakespeare	1595	5	history
5	Titus Andronicus	Shakespeare	(null)	5	tragedy
6	Henry VI part 2	Shakespeare	1592	5	history
7	Richard III	Shakespeare	1593	5	history
8	The Comedy of Errors	Shakespeare	1594	5	comedy
9	Loves Labours Lost	Shakespeare	1594	5	comedy
10	A Midsummer Night's Dream	Shakespeare	1595	5	comedy
11	Romeo and Juliet	Shakespeare	1595	5	tragedy
12	Richard II	Shakespeare	1595	5	history
13	King John	Shakespeare	1596	5	history
14	The Merchant of Venice	Shakespeare	1598	5	comedy
15	Henry IV part 1	Shakespeare	1598	5	history
16	The Merry Wives of Windsor	Shakespeare	1597	5	comedy
17	Henry IV part 2	Shakespeare	1597	5	history
18	Much Ado About Nothing	Shakespeare	1599	5	comedy
19	Henry V	Shakespeare	1599	5	history
20	Julius Ceasar	Shakespeare	1599	5	tragedy

Comporder	Title	Playright	Performed	Acts	Type
21	As You Like It	Shakespeare	1600	5	comedy
22	Hamlet	Shakespeare	1600	5	tragedy
23	Twelfth Night	Shakespeare	1601	5	comedy
24	Troiles Cressida	Shakespeare	(null)	5	comedy
25	Measure for Measure	Shakespeare	(null)	5	comedy
26	Othello	Shakespeare	(null)	5	tragedy
27	Alls Well That Ends Well	Shakespeare	(null)	5	comedy
28	Timon and Athens	Shakespeare	(null)	5	tragedy
29	King Lear	Shakespeare	(null)	5	tragedy
30	Macbeth	Shakespeare	(null)	5	tragedy
31	Anthony and Cleopatra	Shakespeare	(null)	5	tragedy
32	Pericles, Prince of Tyre	Shakespeare	(null)	5	comedy
33	Coriolanus	Shakespeare	(null)	5	tragedy
34	The Winter’s Tale	Shakespeare	(null)	5	comedy
35	Cymbeline	Shakespeare	(null)	5	comedy
36	The Tempest	Shakespeare	(null)	5	comedy
37	Henry VIII	Shakespeare	1613	5	history

Chapter 7: Using the C API

This chapter provides all the information you need to use the Web Deployment Option C API. The API lets you execute any ICE Server function in a remote C application based on GCA. An additional function gives you the ability to download a document from the ICE Server to a remote client.

The Web Deployment Option C API is used primarily to interface with administration software, such as Visual DBA or HTML editing tools.

This chapter provides an alphabetical reference to all of the C API functions and information to help you effectively use the functions. Included are examples showing typical uses of the functions and explanations of how the functions work together.

Note: You need to include the following header file to use the API:

```
#include <ice_c_api.h>
```

Web Deployment Option C API Reference

This section is an alphabetical reference to all of the functions, data structures, and a data type in the Web Deployment Option C API. For a description of some additional data types used in the Web Deployment Option C API, see the *OpenAPI Reference Guide*.

The code examples used throughout this section are taken from the sample Web Deployment Option C API at the end of the chapter. They are explained in the context of that example.

ICE_C_Close() Function

Frees the memory resources created for the row data during fetch operations.

Syntax

ICE_STATUS ICE_C_Close(ICE_C_CLIENT *client*)**;**

Parameters

client — A reference pointer to a structure containing client connection information, returned by the ICE_C_Connect() function.

Returns

SUCCESS if completed successfully; otherwise, an error code indicating the reason for failure.

ICE_C_Connect() Function

Establishes a connection to the ICE Server with the specified node, using the provided user name and password.

Syntax

ICE_STATUS ICE_C_Connect(II_CHAR* *node*, II_CHAR* *user*,
II_CHAR* *password*, ICE_C_CLIENT* *client*);

Parameters

node The name of the vnode associated with the ICE Server machine to which to connect. If a NULL pointer is specified, the local node is assumed.

user The name of the user to connect to the ICE Server.

password The password associated with the user.

client A reference pointer to the client returned by the function. This pointer is used in all subsequent calls to the ICE C API functions.

Important! This reference is used internally by the Web Deployment Option C API interface and should not be modified.

Returns

SUCCESS if completed successfully; otherwise, an error code indicating the reason for failure.

Example

The following line of code establishes a connection to the local node, using the user name and password variables:

```
status = ICE_C_Connect(NULL, username,
    password, &client)
```

The status variable is set to a value of 0 if successful, or a failure code if unsuccessful. Also, the &client variable is assigned the value of a reference pointer to the connection information for the client.

ICE_C_Disconnect() Function

Closes a connection from the connected ICE Server node and cleans memory.

Syntax

ICE_STATUS ICE_C_Disconnect(ICE_C_CLIENT* *client*);

Parameters

client — A reference pointer to a structure containing client connection information, returned by the ICE_C_Connect() function.

Returns

SUCCESS if completed successfully; otherwise, an error code indicating the reason for failure.

ICE_C_Execute() Function

Prepares the server to perform a specified ICE Server extension function.

Syntax

ICE_STATUS ICE_C_Execute(ICE_C_CLIENT *client*,
    II_CHAR* *name*, ICE_C_PARAMS *tab*[]);

Parameters

client — A reference pointer to a structure containing client connection information, returned by the ICE_C_Connect() function.

name — The name of the ICE Server function to execute.

tab[] — An array that is updated with the results of the query, if the query is a select or retrieve operation, and output parameter(s) are specified.

Returns

SUCCESS if completed successfully; otherwise, an error code indicating the reason for failure.

Description

The ICE_C_Execute() function prepares the function for execution by the ICE Server, and then actually executes the query on the database. The result set is stored in memory and can be accessed through the ICE_C_Fetch() and ICE_C_GetAttribute() functions.

An additional Download() function is provided by Web Deployment Option and is accessed through the ICE_C_Execute() function. "download" is specified for *name* and the download parameters are supplied through the *tab*[] array. The document name, business unit name, and target file on the local drive are specified in this array. For example:

```
ICE_C_PARAMS params[] =
{   {ICE_IN, "document", "plays_home.html"},
    {ICE_IN, "ii_unit", "plays"},
    {ICE_IN, "target", local_plays_home},
    {0, NULL, NULL }
};
```

Note: ICE_C_Execute() functions cannot be nested. Each call to the function must be followed by your ICE_C_Fetch(), ICE_C_GetAttribute(), and ICE_C_Close() calls before calling ICE_C_Execute() again.

Example

The following line of code requests that the User() ICE Server function be executed with the parameters specified in the params[] array on the client previously returned from an
ICE_C_ Connect() call:

```
status = ICE_C_Execute(client, "user", params)
```

The params[] array contains the following input and output parameters:

```
ICE_C_PARAMS params[] =
{   {ICE_IN, "action", "select"},
    {ICE_OUT, "user_name", NULL},
    {ICE_OUT, "user_timeout", NULL},
    {0, NULL, NULL }
};
```

A select operation is performed on the user_name and user_timeout columns (properties) for the User() server function. For more information, see the User() Function in the appendix "ICE Server Functions."

The ICE_C_Fetch() and ICE_C_GetAttribute() function descriptions contain examples that detail how the data is accessed.

See Also

ICE_C_PARAMS, ICE_C_Fetch(), ICE_C_GetAttribute()

ICE_C_Fetch() Function

Updates the retrieved row position within the result set obtained by a select or retrieve operation.

Syntax

ICE_STATUS ICE_C_Fetch(ICE_C_CLIENT *client*, II_INT4* *end*);

Parameters

client — A reference pointer to a structure containing client connection information, returned by the ICE_C_Connect() function.

end — An integer returned indicating whether a row is present. If SUCCESS is returned, it indicates that the last row has been referenced and that there are no more rows. If any other value is returned, it means that a row is present.

Description

After a call to the ICE_C_Execute() function, which prepares and executes the query, the ICE_C_Fetch() function updates the retrieved row position. You can access all the data rows by using an if statement and checking on whether the last data row has been reached.

The data rows can be accessed by the ICE_C_GetAttribute() function. See the description of this function to see how the data is manipulated.

Returns

SUCCESS if completed successfully; otherwise, an error code indicating the reason for failure.

Example

In the following code sample, ICE_C_Fetch() updates the retrieved row position within the result set and checks if the row was returned successfully and if it is the last row. (The previous line of code had performed an ICE_C_Execute() function, which defined and executed the select operation.)

```
if ((status = ICE_C_Fetch(client, &end)) == OK && !end)
```

For example, the values for user_id, user_name, user_password1, etc. will be fetched and stored in memory for later retrieval by the ICE_C_GetAttribute() function. for a complete list of the properties that are selected, see the User() Function in the appendix "ICE Server Functions."

See Also

ICE_C_Execute(), ICE_C_GetAttribute()

ICE_C_GetAttribute() Function

Returns the value of the specified server function property within the current row of the select action.

Syntax

```
char* ICE_C_GetAttribute(ICE_C_CLIENT client,
    II_INT4 position);
```

Parameters

client — A reference pointer to a structure containing client connection information, returned by the ICE_C_Connect() function.

position — The number of the ICE_OUT parameter, as specified in the ICE_C_PARAMS array that specifies the server function property name.

Description

The ICE_C_GetAttribute() function is executed after a call to the ICE_C_Fetch() function. It returns the value of the current data row within the column (server function property) specified by the ICE_C_PARAMS array.

The *position* parameter serves as an index into the ICE_C_PARAMS array, and is used to obtain the output parameter name. This is the property name of the server function that was previously executed.

All information returned by Web Deployment Option extension and server functions, including numeric values, is in the form of a character string.

Returns

If successful, a pointer to the text-converted value is returned; otherwise, NULL.

Example

The following line of code prints the values selected by the previously executed ICE_C_Execute() function:

```
printf("%s (%s)\n", ICE_C_GetAttribute(client, 1),
    ICE_C_GetAttribute(client, 2));
```

The ICE_C_GetAttribute() function calls get the data values in the user_name and user_timeout properties columns for the user server function. The "1" and "2" parameters act as indexes into the params[ ] array to extract the "user_name" and "user_timeout" references.

ICE_C_Initialize() Function

Prepares the Web Deployment Option C API for initial use.

Syntax

ICE_STATUS ICE_C_Initialize();

Returns

SUCCESS if completed successfully; otherwise, an error code indicating the reason for failure.

Description

The ICE_C_Initialize() function must be called once before using other functions.

Example

The following code line initializes the C API:

```
ICE_STATUS ICE_C_Initialize();
```

ICE_C_LastError() Function

Returns a textual error message for the last error that occurred.

Syntax

char* ICE_C_LastError(ICE_C_CLIENT *client*);

Parameters

client — A reference pointer to a structure containing client connection information, returned by the ICE_C_Connect() function.

Returns

If successful, a pointer to the textual value is returned; otherwise, NULL, if no message is available.

ICE_STATUS Data Type

Defines a data type for returning status from a Web Deployment Option C API function.

Syntax

typedef II_UINT4 ICE_STATUS;

Description

The ICE_STATUS data type is used by ICE Server functions and its value must be set to NULL initially. If the status becomes a non-NULL (0) value, it indicates an error occurred and an error message must be displayed on the HTML page. To obtain information about the error, use the ICE_C_LastError() function.

Used By

ICE_C_Close(), ICE_C_Connect(), ICE_C_Disconnect(), ICE_C_Execute(), ICE_C_Fetch(), ICE_C_Initialize()

ICE_C_CLIENT Structure

Stores information about a Web Deployment Option connection.

Syntax

typedef II_CHAR* ICE_C_CLIENT;

Description

When you open a connection to the ICE Server using the ICE_C_Connect() function, the ICE_C_CLIENT structure is allocated and a reference pointer is returned. This pointer is used in all calls to the Web Deployment Option C API.

Note: The information is for use internally by the Web Deployment Option C API only and should not be modified by an application.

Used By

ICE_C_Close(), ICE_C_Connect(), ICE_C_Disconnect(), ICE_C_Execute(), ICE_C_Fetch(), ICE_C_GetAttribute(), ICE_C_LastError()

ICE_C_PARAMS Structure

Declares input and/or output parameters for the ICE Server and extension functions.

Syntax

```
typedef struct __ICE_C_PARAMS
{
    II_INT                  type;
    #DEFINE ICE_IN          1
    #DEFINE ICE_OUT         2
    #DEFINE ICE_BLOB        4
    II_CHAR*                name;
    II_CHAR*                value;
} ICE_C_PARAMS;
```

Members

type The type of the parameter represented by an integer expression formed from one or more of the following manifest constants: ICE_IN, ICE_OUT, and ICE_BLOB.

name The name of the parameter.

value The value of the parameter.

Description

This structure is used with ICE Server functions to pass parameter information. The parameters to a server function are passed as an array of ICE_C_PARAMS structures. Each ICE_C_PARAMS entry defines one of the properties of the server or extension function. Depending on the action or request, the parameters must be specified as a combination of either ICE_IN for input, ICE_OUT for output, or ICE_BLOB.

Used By

ICE_C_Execute()

Example

The following code line assigns values to the params[] array, which will be passed to the ICE_C_Execute() function with the name of the "user" server function:

```
ICE_C_PARAMS params[] =
{   {ICE_IN, "action", "select"},
    {ICE_OUT, "user_name", NULL},
    {ICE_OUT, "user_timeout", NULL},
    {0, NULL, NULL }
};
```

The effect will be to choose the action of selecting data from the user_name and user_timeout columns in the Web Deployment Option User() server function. (For the server function properties, see "Appendix D: ICE Server Functions.")

Sample C API for Web Deployment Option

The following example program will select data from the user_name and user_timeout columns within the Web Deployment Option User() server function:

```
#include <stdio.h>
#include <ice_c_api.h>

int
main (int argc, char** argv)
{
ICE_C_CLIENT client = NULL;
ICE_STATUS status = 0;

C_APIInitialize ();

if ((status = ICE_C_Connect(NULL, argv[1], argv[2], &client)) == SUCCESS)
    {
        ICE_C_PARAMS params[] = {       {ICE_IN, "action", "select"},
            {ICE_OUT, "user_name", NULL},
            {ICE_OUT, "user_timeout", NULL},
            {0, NULL, NULL}};
            if ((status = ICE_C_Execute(client, "user", params)) ==
                SUCCESS)
            {
                int end;
                do
                {
                        if ((status = ICE_C_Fetch(client, &end)) ==
                                SUCCESS && !end)
                                        printf("%s (%s)\n",
ICE_C_GetAttribute(client, 1),
                                        ICE_C_GetAttribute(client, 2));
                }
                while (status == 0 && !end);
                ICE_C_Close(client);
            }
            ICE_C_Disconnect(&client);
        }
return(status);
}
```

Chapter 8: Writing ICE Server Extension Functions

You can write C-callable ICE Server extension functions that can be invoked from an HTML macro page. A strict interface enables the ICE Server to construct a list of function names and their parameter names from the loaded module (DLL or shared library) and to pass values to and retrieve values from an extension function.

This chapter examines the concepts used when writing server extension functions and calling them from within your applications. The examples used throughout the chapter are taken from the sample Plays tutorial Web application.

Defining an Initialization Function

An initialization function called InitICEServerExtension() must be present within your extension library in order to provide a description of the extension function. It serves as an entry point to determine all other entry points. This function provides the address of a structure that holds the function description**.**

The InitICEServerExtension() function is specified as follows:

char* InitICEServerExtension(PSERVER_DLL_FUNCTION*
ppServerDllFunction);

The *ppServerDllFunction* parameter is a pointer that is updated with the address of the structure containing a function description table. The section that follows discusses the specification of this structure.

The InitICEServerExtension() must return NULL if it is completed successfully. If unsuccessful, the function must return an error string that is displayed via the HTML page.

Example

The following initialization function is used in the play_neworder.sc file in the src subdirectory under the plays directory. It returns a pointer to the FunctionTable structure:

```
InitICEServerExtension(PSERVER_DLL_FUNCTION*     ppServerDllFunction)
{
    ICE_STATUS status= NULL;
    *ppServerDllFunction = FunctionTable;
    return status;
}
```

Providing a Function Description

In your extension library, you must provide a structure that describes the function. It defines two members describing the function name and its parameters. The structure is specified as follows:

typedef struct *structure_name*
{
    char* *pszName***;**
    char** *pszParams***;**
}SERVER_DLL_FUNCTION, *PSERVER_DLL_FUNCTION

The first member shown, *pszName*, is a string pointer that points to a string providing the function name. The function name must be identical to the name that is declared in the code within your HTML page.

The *pszParams* member is an array of string pointers that point to the list of parameter name strings. These names are used to ensure that parameters from the macro are passed in the correct order to the function.

Example

This example shows how the newOrder extension function is described within the play_newOrder library, using a structure and several static variable definitions. Within the play_newOrder.sc file, the ice_function_table structure appears as follows:

```
typedef struct ice_function_table
{
    char*       pszName;
    char**      pszParams;
}SERVER_DLL_FUNCTION, *PSERVER_DLL_FUNCTION;
```

Now let us see how the FunctionTable[] array is assigned. The function name is newOrder and the pointer to the parameter name list is pszNewOrderParams:

```
static SERVER_DLL_FUNCTION FunctionTable[] =
{
    { "newOrder", { pszNewOrderParams } },
    { NULL }
};
```

The pszNewOrderParams pointer was assigned in a prior line in the file as follows:

```
static char* pszNewOrderParams[] =
    {"out_orderNumber", NULL};
```

In effect, the newOrder function will be called and the value of out_orderNumber will be returned to the program through a reference to ":orderNumber" in the HTML code. We will look at how the newOrder function works in the next section.

Defining Your Extension Function

An ICE Server extension function provides the initialization and the termination processing for the function in addition to performing the extension function.

ICE Server Extension Function

You specify a server extension function, *extension_name,* as follows:

char* *extension_name*(**char**** *pszParam*, **BOOL*** *bData*,
char** *pContext*)

The parameters specified for an extension function are described below:

Parameter	Description
pszParam	A list of parameters used for input and output values. An empty string in the first input parameter indicates the end of the input parameters.
bData	Flag indicating validity of parameter values on output. Only used as an output parameter, TRUE indicates output parameters are valid for display. FALSE (default) indicates that no output processing is required after return.
pContext	A private storage area used by the function and should not be modified by the caller. On the first invocation of the function, during which initialization must be performed, the context pointer is updated with the context storage area. This is for exclusive use of the function and should not be modified by the caller.

Termination Processing

To signal termination, the ICE Server calls the function with an empty string for the first input parameter after all values have been processed. This indicates to the function to free all allocated memory.

Since all extension functions run in the context of the ICE Server, it is imperative that they are thread-safe and execute as efficiently as possible.

Example

In this example, the newOrder() extension function is shown. It calls an Ingres procedure (new_order) to generate an order for an Internet shopping application. The procedure sets the out_OrderNumber parameter, which is passed back to the calling Web page, play_shopHome.html.

The newOrder() function appears as follows:

```
newOrder(char** out_OrderNumber, BOOL* print, char**      context)
{
```

```
    ICE_STATUS  status = NULL;
    *print = FALSE;
    /*
** If the context is NULL this is the first invocation.
** Declare an SQL variable to take the returned value from the
** procedure and allocate memory for returning the ice result.
**
** Connect to the icetutor database, execute the procedure and
** store the value.
**
** Set the print flag to indicate valid data returned.
*/
    if (*context == NULL)
    {
        exec sql begin declare section;
        long x;
        exec sql end declare section;

        *context = HeapAlloc(GetProcessHeap(),
            HEAP_ZERO_MEMORY, MAX_SIZE);
        if (*context == NULL)
        {
            return ("Memory error\n");
        }

        exec sql connect 'icetutor' identified by
            'icedbuser';
        exec sql execute procedure new_order into :x;

        exec sql commit;
        exec sql disconnect;

        sprintf (*context, "%d", x);

        *out_OrderNumber = *context;
        *print = TRUE;
    }
    else
    {
        HeapFree (GetProcessHeap(), 0, *context);
        *context = NULL;
    }

    return (status);
}
```

Calling an Extension Function from a Web Page

For an example of how extensions are called from within an application, we will now take a look at the play_shopHome.html page in the Plays application. A Web Deployment Option macro uses the FUNCTION macro keyword to call the newOrder extension function within the play_NewOrder library.

The Web Deployment Option macro appears as follows:

```
<!-- #ICE REPEAT IF (DEFINED (e_orderNumber) AND
    `:e_shopTxn` != `COMPLETE`)
        THEN=``
        ELSE=`<!-- #ICE REPEAT
            FUNCTION=``play_NewOrder.newOrder``
            HTML=``<!-- #ICE
                DECLARE=````session.e_orderNumber=
```

```
                :out_orderNumber````
                -->``
        -->`
-->
```

The HTML keyword is then used to display the value resulting from the function call, which is the order number, represented by the out_orderNumber variable.

For a description the FUNCTION macro keyword, see FUNCTION Keyword in the chapter "Using the Macro Language."

Sample Extension Library

For your reference, this section contains an example ICE Server extension library—including an initialization function, extension function, and supporting code.

Plays Example

Windows

The header file and extension source code file for play_newOrder is shown below.

play_NewOrder.h

The play_newOrder.h header file appears as follows:

```
# include <windows.h>

# define ICE_EXT_API declspec(dllexport)

typedef char*            ICE_STATUS;
typedef ICE_STATUS       (*PFNEXTENSION) (char**, BOOL*,  char **);

typedef struct ice_function_table
{
    char*       pszName;
    char**      pszParams;
}SERVER_DLL_FUNCTION, *PSERVER_DLL_FUNCTION;

typedef ICE_STATUS       (*PFNINITIALIZE)(PSERVER_DLL_FUNCTION*);
```

play_newOrder.sc

The play_newOrder.sc extension source code file appears as follows:

```
# include "play_NewOrder.h"

# defineMAX_SIZE20  /* about the right size
    for an int */
/**
** Parameter name list.
** A NULL pointer terminates the list.
```

```
*/
static char* pszNewOrderParams[] =
    {"out_orderNumber", NULL};

/**
** Function Description
*/
static SERVER_DLL_FUNCTION FunctionTable[] =
{
        { "newOrder", { pszNewOrderParams } },
        { NULL }
};

/*
 * Name: InitICEServerExtension
 *
 * Description:
 *      Mandatory function for providing function
 *      description to the server.
 *
 * Inputs:
 *      None.
 *
 * Outputs:
 *      ppServerDllFunction: pointer to the function
 *      description structure.
 *
 * Returns:
 *      pointer to error text
 *      NULL on success
 */
ICE_EXT_API ICE_STATUS
InitICEServerExtension(PSERVER_DLL_FUNCTION*
    ppServerDllFunction)
{
    ICE_STATUS status= NULL;
    *ppServerDllFunction = FunctionTable;
    return status;
}

/*
 * Name: newOrder
 *
 * Description:
 *      Return the next order number
 *
 * Inputs:
 *      None.
 *
 * Outputs:
 *      out_OrderNumber
 *
 * Return:
 *      pointer to error text
 *      NULL on success
 */
ICE_EXT_API ICE_STATUS
newOrder(char** out_OrderNumber, BOOL* print, char**
    context)
{
    ICE_STATUS  status = NULL;

    *print = FALSE;
    /*
    ** if first invocation allocate some memory for
    ** the result
    */
    if (*context == NULL)
```

```
    {
        exec sql begin declare section;
        long x;
        exec sql end declare section;

    *context = HeapAlloc(GetProcessHeap(),
        HEAP_ZERO_MEMORY, MAX_SIZE);
    if (*context == NULL)
    {
        return ("Memory error\n");
    }

    exec sql connect 'icetutor' identified by
        'icedbuser';
    exec sql execute procedure new_order into :x;

    exec sql commit;
    exec sql disconnect;

    sprintf (*context, "%d", x);

    *out_OrderNumber = *context;
    *print = TRUE;
    }
    else
    {
        HeapFree (GetProcessHeap(), 0, *context);
        *context = NULL;
    }

    return (status);
}
```

Appendix A: XML Primer

This appendix provides an introduction to XML (Extensible Markup Language), some basic XML syntax, and examples of how the Web Deployment Option macro language can be used to create XML applications.

XML Overview

XML is a standard that allows users to specify their own tags. Tags can be nested, just like in HTML. This means that XML can be used to describe any form of data that conforms, or can be transformed to conform, to a tree structure. XML is a markup language much like HTML—in fact they have a common source: SGML. HTML was designed primarily to describe how to present data whereas XML was designed to describe the data itself. Unlike HTML, tags are not predefined in XML; the user must define their own. Currently users formally describe their tags using a DTD (Document Type Definition).

XML does not replace HTML; the two have different and complementary goals. XML was designed to describe data and HTML was designed to describe how to display data.

Tip: For a complete description of the latest XML standard, you can access the URL http://www.xml.org.

Extensible

In HTML, the tags that can be used and their structure (the order in which they can appear) are fixed by the HTML standard. Only by changing the standard can new tags be introduced. The only tags an HTML author can use are those that are defined by the standard. In contrast, in XML the author first defines their own tags and structure, and then they create their documents. The tag set can thus be fine-tuned to a particular problem domain.

Complementary with HTML

It is worth pointing out that XML should not be seen as a replacement for HTML. It seems highly likely that XML will be increasingly used to exchange data between computer system, including between a web server and its clients. When the data is to be presented to a browser, it will be transformed into something that the browser is designed to accept, for example HTML or WML.

XML Syntax

This section presents some basic XML syntax rules.

All Elements Have A Closing Tag

In HTML some elements do not have to have a closing tag. For example, the following code is legal in HTML:

```
<p>This is my first paragraph
<p>This is my second paragraph
```

In XML all elements ***must*** have a closing tag like this:

```
<p>This is my first paragraph</p>
<p>This is my second paragraph</p>
```

XML Tags Are Case Sensitive

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>. Opening and closing tags must therefore be written with the same case:

```
<Letter>This is incorrect</letter>
```

```
<letter>This is correct</letter>
```

XML Elements Must Be Properly Nested

In HTML it is possible to have a construct where the elements intersect one another, for example:

```
<b><i>bold and italic text</b></i>
```

In XML all elements must be properly nested within each other like this:

```
<b><i>bold and italic text</i></b>
```

XML Documents Must Have a Root Tag

All XML documents must contain a single start/end tag pair to define the root element. All other elements must be nested within the root element. Any element can contain sub (child) elements. The sub elements must be in pairs and correctly nested within their parent element:

```
<root_element>
    <child_element>
        <subchild>
            Sub child data
        </subchild>
    </child_element>
</root_element>
```

Attribute Values Must Always Be In Quotation Marks

As in HTML, XML elements can have attributes. Unlike in HTML, in XML, the attribute values must always be surrounded by quotation marks.

XML Example

Memo Example

Here is an example of a simple, but complete, XML document.

```
<?xml version="1.0"?>
<memo>
    <to>Fred</to>
    <from>Harriet</from>
    <subject>Weekend</subject>
    <memoBody>Would you like to go Yachting on Saturday?</memoBody>
</memo>
```

The first line in the document contains the XML declaration, which should always be included. It defines the XML version of the document. In this case, the document conforms to the 1.0 specification of XML:

```
<?xml version="1.0"?>
```

The next line defines the first element of the document (the root element):

```
<memo>
```

The following lines defines 4 child elements of the root (to, from, Subject, and memoBody):

```
<to>Fred</to>
<from>Harriet</from>
<Subject>Weekend</Subject>
<memoBody>Would you like to go Yachting on Saturday?</memoBody>
```

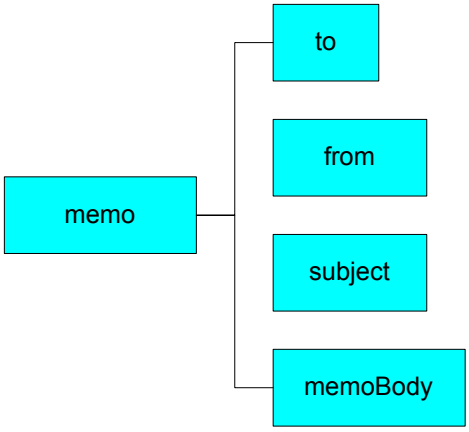
Finally, the last line defines the end of the root element:

```
</memo>
```

Note: The elements simply describe the data. There is no display-related information. Display information could be added.

Memo DTD

The following diagram represents the DTD used to validate the above example



Here is the same DTD in text form:

```
<!ELEMENT memo  (to , from , subject , memoBody )>
<!ELEMENT to  (#PCDATA )>
<!ELEMENT from  (#PCDATA )>
<!ELEMENT subject  (#PCDATA )>
<!ELEMENT memoBody  (#PCDATA )>
```

XML and Web Deployment Option Queries

An XML vocabulary has been developed to work with Web Deployment Option and this has been appended to the XHTML standard. A translation tool is provided to process the XML ICE language into a form that Web Deployment Option can interpret. The result of this is that a Web author can use any XML-aware editor to create Web Deployment Option templates. Here is an example snippet of code generated with one such tool:

```
<i3ce_query i3ce_database="shakespeare">
    <i3ce_sql i3ce_transaction="MyTransaction“>
        <i3ce_statement>
            select title, type from plays
        </i3ce_statement>
        <i3ce_headers>
            <i3ce_header i3ce_column="title" i3ce_text="Play Title"/>
            <i3ce_header i3ce_column="type" i3ce_text="Type of Play"/>
        </i3ce_headers>
        <i3ce_relation_display i3ce_typename="i3ce_table"/>
    </i3ce_sql>
</i3ce_query>
```

Appendix B: HTML Primer

HTML is a collection of styles (indicated by markup tags) that defines the components of a Web document. HTML documents are in plain text format (also known as ASCII) and can be created using any text editor.

This appendix uses some brief examples to describe the small subset of the HTML standard that is used by Web Deployment Option and in the examples in this guide.

The Development of HTML

HTML was originally developed by Tim Berners-Lee while at CERN. During the 1990s, while the Web experienced explosive growth, HTML also grew beyond its simple beginnings. The original aim of HTML was to enable research workers to share their papers electronically. For this to happen, it was sufficient to establish a one-time connection between the browser and the server to download the required information and display it to the user. Each connection existed on its own and was not related to any other. There was no session information and the connections were therefore stateless. This lead to problems, especially in the field of database access and transaction control. These days, state information is typically held in the form of a so-called "cookie," which is sent by the server to the client.

In all, there are less than 100 tags in the HTML standard; however, producing acceptable results using Web Deployment Option does not require knowledge of all of them. In this appendix, only the most important HTML tags from a Web Deployment Option point of view are presented.

Tip: For a complete description of the HTML 4.01 standard, you can access the URL http://www.w3.org/TR/html401/.

Anatomy of an HTML Document

To begin, we will take a quick tour of the structure of an HTML document. An HTML document is made up of two types of content. These are the text itself and formatting instructions known as *markup*. Markup *tags* may, or may not, surround text. The combination of markup tags and their associated text is known as an *element*.

General Usage of HTML Elements

For example, a level three section heading would be indicated as follows:

```
<H3>
This is a level three section heading
</H3>
```

In this example of a level three heading element, the heading is introduced by the tag <H3>. The text of the heading follows and is terminated by the </H3> tag. The example could equally well be shown as follows:

```
<H3> This is a level three section heading </H3>
```

The new lines can be included for clarity (in HTML, new lines count as white space). As we can see from this example, text appears as is, without any decoration.

The HTML tags are distinguished from normal text by angle brackets "<" and ">". For example, the
 tag denotes a line break. HTML makes no distinction about the case of the tags, so the line break tag could also appear as
.

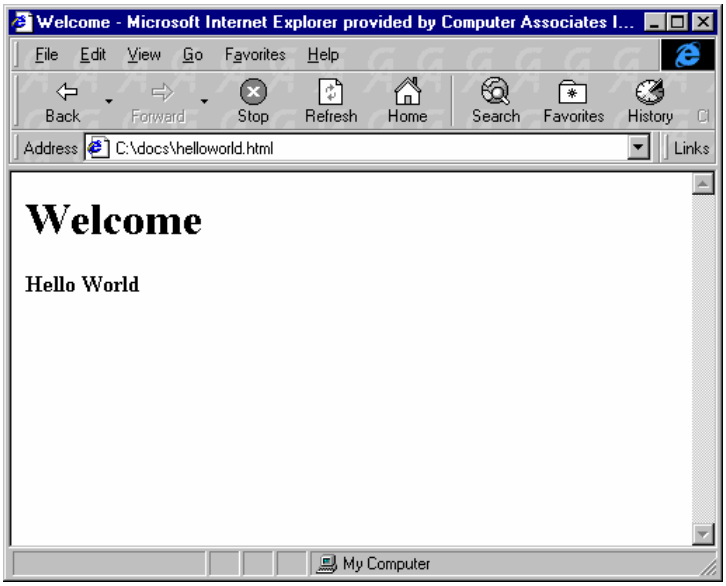
Document Header and Body

Generally, an HTML document consists of two parts: a header and a body. A limited number of elements are allowed in the header, the rest appear in the body. Comments are allowed in both. Here is a simple example of a simple HTML page:

```
<HTML>
<HEAD>
<TITLE>
Welcome
</TITLE>
</HEAD>
<BODY>
<H1>
Welcome
</H1>
<!-- This is a comment -->
<B>Hello World</B>
</BODY>
</HTML>
```

Displaying the HTML Document in a Browser

If displayed in a browser, this simple HTML document would appear as follows with our main message "Hello World":



Let's examine the individual elements in the document. Note that most tags come in pairs, though there are notable exceptions (for example, line break, denoted by
).

Document Beginning and End Tags

In this example, the entire document is bounded by the <HTML> tag pair:

```
<HTML>
...
</HTML>
```

Header

Following the <HTML> tag is the *head* element, bounded by the tags:

```
<HEAD>
...
</HEAD>
```

Title

The head element contains a *title* element that appears as follows:

```
<TITLE>
Welcome
</TITLE>
```

The title typically appears in the title bar of the browser window.

Body

The head element is followed by the main *body* element of the document. This is bounded by the tags:

```
<BODY>
...
</BODY>
```

Headings

The body element can contain a large number of other elements—here it contains a heading and some plain text. Traditionally, the level one heading of a document is identical to its title. The level one heading in our example is as follows:

```
<H1>
Welcome
</H1>
```

Comments

The heading is followed by an HTML comment:

```
<!-- This is a comment -->
```

The comment is not rendered by the browser and is made use of extensively by Web Deployment Option.

Document Output

Our main message, "Hello World," appears as output in a browser.

HTML Rendering and Code Readability

It is worth remembering that the HTML browser is responsible for formatting the text according only to the directives laid down by the HTML tags accompanying the text. In particular, new lines count as white space. Therefore, a browser will render a very long source text line in the same way as it would a source text with one word per line. The text of our example could equally well have appeared as follows:

```
<B>
Hello
World
</B>
```

It would have been displayed exactly as before. To preserve readability in HTML files, headings should be placed on separate lines and blank lines (in addition to the <P> tags) should separate paragraphs.

The body element is terminated with the </BODY> tag and the entire document is terminated with the </HTML> tag.

It is worthwhile spending a couple of minutes typing this example in with a text editor and then checking that the rendered result in a browser looks like that depicted previously in this appendix.

Elements Used by Web Deployment Option

An HTML document can be made dynamic—that is, so that its content is updated automatically to reflect the contents of the database by passing requests to the ICE Server. This is typically achieved by submitting the request as an HTML *form*. Parameters can be passed by setting Web Deployment Option variables in the form.

An HTML form is an element containing, in addition to the normal markup elements, special elements called *controls*. These controls allow a user to fill in a form prior to submitting it for processing by, for example, Web Deployment Option through a web server. The following code sample produces a dynamic SQL input form:

```
<HTML>
<HEAD>
<TITLE>Web Deployment Option Dynamic SQL Demo</TITLE>
</HEAD>
<BODY>
<FORM ACTION="/bin/oiice.exe" METHOD="GET">
<H1>
Web Deployment Option Dynamic SQL Console
</H1>
User Id: <INPUT TYPE=text NAME="ii_userid" MAXLENGTH=32 VALUE="ingres">
Password: <INPUT TYPE=password NAME="ii_password">
<HR>
<INPUT TYPE=hidden NAME="ii_database" VALUE="icedb">
<TEXTAREA ROWS=6 COLS=80 NAME="ii_query_statement">
select * from <my_table>
</TEXTAREA>

<INPUT TYPE=SUBMIT VALUE="Execute Query">
<INPUT TYPE=RESET VALUE="Start Again">
</FORM>
</BODY>
</HTML>
```

In this example, the HEAD section is essentially the same as in the previous one. We will therefore only examine the new features. This is the form that appears in the body of the document. The following tag introduces the form element:

```
<FORM ACTION="/bin/oiice.exe" METHOD="GET"
    TARGET="SQL_OUT">
```

A form provides a way of gathering user input. This form tag has two *attributes*. The first, ACTION, specifies the entity responsible for dealing with the form once it has been submitted. In this case, a binary executable, /bin/oiice.exe, processes the data submitted. The second attribute, METHOD, denotes how the data will be presented to the oiice.exe program. The GET method appends the form data set to the form URI (specified by the ACTION attribute) and the new URI is sent upon submission. The POST method includes the data set in the body of the form.

- The Get() method is used when the form has no side effects. For example, database searches have no visible side effects and make ideal applications for the Get() method.

- If the service associated with the processing of a form causes side effects (for example, if the form updates a database), the Post() method should be used.

Following the level one heading "Web Deployment Option Dynamic SQL Demo" the next new feature is apparent, an input control:

```
<INPUT TYPE=text NAME="ii_userid" MAXLENGTH=32
    VALUE="ingres">
```

This INPUT tag specifies a default (text) input control, which accepts one line of input limited to a maximum length of 32 characters with the MAXLENGTH attribute. The text the user types in is associated with the named variable ii_userid and a default value of ingres is supplied. The next input control is of a different type, namely "password":

```
<INPUT TYPE=password NAME="ii_password">
```

The control is basically the same as for "text", but the user input is rendered in such a way as to obscure what the user typed (e.g., by echoing a series of asterisks). Note that the password text is sent over the network as clear text.

Another type of input control, the HIDDEN type, appears next:

```
<INPUT TYPE=hidden NAME="ii_database"
    VALUE="icedb">
```

Here the author wants to set a variable but not render it, hiding the value from the user. In this case, the ii_database variable contains the value "icedb." The user sees nothing on the rendered page. Following the hidden input control is an element that allows more than one line of input, an area of text:

```
<TEXTAREA ROWS=6 COLS=80 NAME="ii_query_statement">
    select * from <my_table> </TEXTAREA>
```

The <TEXTAREA> tag specifies the size of the visible input area, here 6 rows by 80 columns (if the user needs more space, the browser scrolls) and the variable is named ii_query_statement. An initial value is set to be "select * from <my_table>", the angle brackets indicating that the user should supply a table name to complete the statement. The element is completed by a mandatory </TEXTAREA> tag.

Finally, the form is completed by having a SUBMIT and a RESET button. These appears as:

```
<INPUT TYPE=SUBMIT VALUE="Execute Query">
<INPUT TYPE=CANCEL VALUE="Start Again">
```

The SUBMIT control causes the form's data set (that is, the control name-value pairs) to be submitted to the oiice.exe program.

The RESET control causes the controls on the form to be reset to their initial values; if a control does not have an initial value, the effect of a reset is not defined.

The form element is terminated by the </FORM> tag.

An input form is used to gather user input and to act upon it. In this example, we have seen how to mimic the action of the Terminal Monitor in a browser, but we could just as easily have run a report or executed a database procedure. Other chapters in this guide deal with these topics; the basics of the HTML remain the same.

Elements Generated by Web Deployment Option

To reduce the burden on the Web author still further, Web Deployment Option can generate some popular HTML elements automatically. These include the following:

- Table
- Selector control
- Parameterized hyperlink

In addition, Web Deployment Option can generate all the HTML tags required to present a standard report (for example, generated by RBF) to a Web browser. It wraps the report on your behalf. The result is "What You Got Is What You See" (WYGIWYS).

The Web Deployment Option macro processor generates the table, selector control, and parameterized hyperlink elements. For more information, see "**Chapter 6: Creating Web Applications:** An Example."

Accessing Web Deployment Option Pages

Finally, you can access a Web Deployment Option document through an address that is formatted with the following syntax.

Windows

On Windows, the syntax is:

/ice-bin/oiice.[**dll**|**exe**]**/**[*session_group/*][*business_unit***[**]{*location_name/*}*document_name*[**]**]

For example:

```
/ice-bin/oiice.dll/mygroup/myunit[myloc/mydoc.html]
```

UNIX

On UNIX, the syntax is:

/ice-bin/oiice.1.so/[*session_group/*][*business_unit***[**]{*location_name/*}*document_name*[**]**]

For example:

```
/ice-bin/oiice.1.so/mygroup/myunit[myloc/mydoc.html]
```

Appendix C: Reserved Words

HTML variables are defined using the HTML <INPUT> tag. A Web Deployment Option reserved word and corresponding value can be specified for the NAME and VALUE options, as shown below:

```
<INPUT TYPE="input_type" NAME="ice_reserved_word" VALUE="value">
```

Reserved Words

The following table summarizes the HTML variables Web Deployment Option recognizes. The reserved words provided with Web Deployment Option (formerly Ingres/ICE) Version 2.0 are supported for compatibility purposes. When applicable, you should use Version 2.5 reserved words because they provide enhanced functionality.

Reserved Word	Description
http_remote_addr (2.5)	The IP address of the requestor.
http_remote_host (2.5)	The resolved name of the requestor. If DNS resolution is not available, the value will be the same as http_remote_addr.
http_user_agent (2.5)	The browser type of the requestor.
ii_action (2.5)	The action to be taken when opening or closing a Web Deployment Option session. Valid values are: declare—used to create a new Web user with a specified user name, password, and profile (using the ii_userid, ii_password, and ii_profile variables). connect—used to open a session with the ICE Server with a specified user and password (using the ii_userid and ii_password variables). disconnect—used to close a session with the ICE Server.
ii_application	The user application to execute in the web server's CGI directory.
ii_authtype (2.5)	The password authentication type. Possible values are ICE, for Web Deployment Option authentication, and OS, for operating system authentication.

Reserved Word	Description
ii_binary_ext	The default file extension to use when storing a BLOB from a database to disk. This extension is used when Web Deployment Option is unable to determine the file type stored in a database.
ii_cookie (2.5)	The cookie used to maintain session context on variable-based pages.
ii_database	The name of the database used when executing a report or an SQL-based request. If a value for ii_database is not defined, an error is generated and returned to the client.
ii_error_message (2.5)	The message to be displayed to the client when a Web Deployment Option request has not completed successfully. If the Web author does not provide a value, a default error message is displayed. It is recommended that a value for ii_error_message be assigned so that clients do not become confused as to why a particular error message (an internal error message, for example) has appeared. To display this message to the Web client, ii_error_url must not be defined.
ii_error_url (2.5)	The HTML page to be loaded by the web server if an Ingres request has not completed successfully. If this is not defined, the value of ii_error_message is displayed.
ii_output_dir	The label of the directory to be used for Web Deployment Option file output.
ii_page_header	The Report-Writer output page heading. The value for this variable is equivalent to the .HEADER page report structure statement. It is printed at the top of the report. This variable is used when processing report specifications that do not have HTML tags embedded in them.

Reserved Word	Description
ii_password	The password for a Web user. If no user or password is supplied, the product attempts to use HTTP basic authentication, if enabled on the web server; otherwise, the default user alias stored in the configuration file is used. For Version 2.0, ii_password and ii_userid are used to verify the user executing a request. A value for ii_password is required only if ii_userid is supplied.
ii_procedure	The name of the database procedure to be executed by Web Deployment Option. This is a more secure way to execute SQL statements and is recommended if you plan on having your Ingres installation accessible by the general public. If you are passing variables to a procedure, ii_procvar_count must be defined and the number of variable names, data types, and lengths must equal the number of variables that are being passed to the procedure.
ii_procvar_count	The number of variables that are going to be passed into the procedure to be executed, which is defined in ii_procedure.
ii_procvar_data*n*	The data that is to be assigned to a procedure variable name (defined in ii_procvar_name*n*) when it is passed to a database procedure. The letter *n* represents the data item that corresponds to each procedure variable name (such as ii_procvar_data1, ii_procvar_data2, etc.). The number of data items must equal the value assigned to ii_procvar_count.
ii_procvar_length*n*	The procedure variable length to be passed to a database procedure. The letter *n* represents the procedure variable length number (such as, ii_procvar_length1, ii_procvar_length2, etc.). The number of procedure variable lengths must equal the value assigned to ii_procvar_count.

Reserved Word	Description
ii_procvar_name*n*	The name of the procedure variable to be passed to a database procedure. The letter *n* represents the procedure variable number (such as, ii_procvar_name1, ii_procvar_name2, etc.). The number of procedure variable names must equal the value assigned to ii_procvar_count.
ii_procvar_type*n*	The procedure variable data type to be passed to a database procedure. The letter *n* represents the procedure variable data type number (such as, ii_procvar_type1, ii_procvar_type2, etc.). The number of procedure variable data types must equal the value assigned to ii_procvar_count. Valid data types are: integer1 integer2 integer4 float4 float8 byte varchar char long byte long varchar text
ii_profile (2.5)	The name of a profile for a newly created user (created by setting ii_action=`declare`). This gives the user privileges to access the remaining documents.
ii_query_statement	The dynamic SQL statement to be executed by Web Deployment Option. Variables provided for ii_query_statement are case-sensitive. The following example indicates to Web Deployment Option that there are two *different* variable names being used in the query statement, value1 and VALUE1: insert into table1 values (:value1, :VALUE1);

Reserved Word	Description
ii_report	The report to be executed by the Report Writer utility. If the report name stored in ii_report does not exist in the database, an error is returned to the client. If ii_report is defined, ii_report_location is ignored.
ii_report_header	The Report-Writer output report heading. The value for this variable is equivalent to the .HEADER report structure statement. It becomes the title of the HTML page. This variable is used when processing report specifications that do not have HTML tags embedded in them.
ii_report_location	The full path and file name of the report specification to use when generating a report. When using this variable, the report specification does not have to be stored in the database. If the report specification defined in ii_report_location does not exist, an error is returned to the client.
ii_rowcount (2.5)	The row count of the last SQL query.
ii_rwdir	The directory in which to store Report-Writer output. This variable is supported for backward compatibility for 2.0 only. It has been replaced by ii_output_dir.
ii_status_info (2.5)	Additional information associated with the last error.
ii_status_number (2.5)	The status number of the last executed macro.
ii_status_text (2.5)	The text associated with the status number.
ii_success_message (2.5)	The message to be displayed to a Web client when a Web Deployment Option request has completed successfully. This variable is ignored if ii_success_url has been defined. If a value for ii_success_message is not supplied, a default message is returned to the client.
ii_success_url (2.5)	The URL to be loaded by the web server if an Ingres request has completed successfully.
ii_system	The location of the Ingres installation. If the location is not defined, an attempt is made to retrieve the value from the environment settings.

Reserved Word	Description
ii_unit (2.5)	The business unit associated with the active session.
ii_userid	The name of a Web user (an aliased name). If no user or password is supplied, basic authentication is used (if enabled on the web server); otherwise, the default user alias stored in the configuration file is used. For Version 2.0, ii_userid and ii_password are used to verify the user executing a request.

Appendix D: ICE Server Functions

This appendix provides the properties and actions that are associated with each of the available ICE Server functions. Using these functions, you can write an application that accesses, manages, and monitors all the information contained in the Web Deployment Option repository.

Security Functions

The functions in this section allow you to access the security information in Web Deployment Option, which pertains to database users, database connections, roles, Web users, profiles, and the associations between some of these objects.

DBUser() Function

Provides access to the properties of one or more database user(s).

The properties and actions for the DBUser() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	4	4	4	4	
dbuser_id	out	in	in	in	in	Unique database user identifier
dbuser_name	out	out	in	in	none	Database user name
dbuser_alias	out	out	in	in	none	Web Deployment Option name for database user
dbuser_password1	out	out	in	in	none	Database user password
dbuser_password2	out	out	in	in	none	Database user password confirmation
dbuser_comment	out	out	in	in	none	Comment for database user

Database() Function

Provides access to the properties of a database connection.

The properties and actions for the Database() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	4	4	4	4	
db_id	out	in	in	in	in	Unique database connection identifier
db_name	out	out	in	in	none	Virtual database name
db_dbname	out	out	in	in	none	Actual database name
db_dbuser	out	out	in	in	none	Unique identifier for database user
db_comment	out	out	in	in	none	Comment for database connection

Role() Function

Provides access to the properties of a role.

The properties and actions for the Role() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	4	4	4	4	
role_id	out	in	in	in	in	Unique role identifier
role_name	out	out	in	in	none	Role name
role_comment	out	out	in	in	none	Comment for role

User() Function

Provides access to the properties of a Web user.

The properties and actions for the User() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	4	4	4	4	
user_id	out	in	in	in	in	Unique Web user identifier
user_name	out	out	in	in	none	Web user name
user_authtype	out	out	in	in	none	Authentication type (default = ICE)
user_password1	out	out	in	in	none	Web user password
user_password2	none	none	in	in	none	Web user password confirmation
user_dbuser	out	out	in	in	none	Unique database user identifier
user_comment	out	out	in	in	none	Comment
user_administration	out	out	in	in	none	Member of administrators; enabled if TRUE
user_security	out	out	in	in	none	Member of security managers; enabled if TRUE
user_unit	out	out	in	in	none	Member of business unit managers; enabled if TRUE
user_monitor	out	out	in	in	none	Allowed to monitor; enabled if TRUE
user_timeout	out	out	in	in	none	The maximum idle time, in seconds, between requests to the server

User_Role() Function

Provides access to the properties for a role associated with a Web user.

The properties and actions for the User_Role() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	7	4	7	4	
ur_user_id	out	none	in	none	in	Unique Web user identifier
ur_role_id	out	none	in	none	none	Unique identifier of role associated with Web user
ur_role_name	out	none	none	none	none	Name of role associated with Web

	user

User_Database() Function

Provides access to the properties for a database connection associated with a Web user.

The properties and actions for the User_Database() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	7	4	7	4	
ud_user_id	in	none	in	none	in	Unique user identifier
ud_db_id	out	none	in	none	none	Unique identifier of database connection associated with user
ud_db_name	out	none	none	none	none	Name of database connection associated with user

Profile() Function

Provides access to the properties of a profile.

The properties and actions for the Profile() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	4	4	4	4	
profile_id	out	in	in	in	in	Unique profile identifier
profile_name	out	out	in	in	none	Profile name
profile_dbuser	out	out	in	in	none	Unique database user identification
profile_administration	out	out	in	in	none	Member of administrators; enabled if TRUE
profile_security	out	out	in	in	none	Member of security managers; enabled if TRUE
profile_unit	out	out	in	in	none	Member of business unit

						managers; enabled if TRUE
profile_monitor	out	out	in	in	none	Allowed to monitor; enabled if TRUE
profile_timeout	out	out	in	in	none	The maximum idle time, in seconds, between requests to the server

Profile_Role() Function

Provides access to the properties for a role associated with a profile.

The properties and actions for the Profile_Role() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	7	4	7	4	
pr_profile_id	out	none	in	none	in	Unique profile identifier
pr_role_id	out	none	in	none	none	Unique identifier of role associated with profile
pr_role_name	out	none	none	none	none	Name of role associated with profile

Profile_Database() Function

Provides access to the properties for a database connection associated with a profile.

The properties and actions for the Profile_Database() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	7	4	7	4	
pd_profile_id	in	none	in	none	in	Unique profile identifier
pd_db_id	out	none	in	none	none	Unique database identifier
pd_db_name	out	none	none	none	none	Database name

Business Unit Functions

The functions in this section allow you to access the business unit information within Web Deployment Option, which pertains to business units, documents, session groups, and the associations between some of these objects and others such as roles, Web users, and locations. In addition, a function is provided that allows you to back up your business unit.

Unit() Function

Provides access to the properties of a business unit.

The properties and actions for the Unit() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	4	4	4	4	
unit_id	out	in	in	in	in	Unique business unit identifier
unit_name	out	out	in	in	none	Virtual business unit name
unit_owner	out	out	none	none	none	Actual business unit name

Unit_Role() Function

Provides access to the properties of a role access definition for a business unit.

The properties and actions for the Unit_Role() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	7	4	7	4	7	
ur_unit_id	none	in	none	in	none	Unique business unit identifier
ur_role_id	none	out	none	in	none	Unique identifier for role associated with business unit
ur_role_name	none	out	none	none	none	Name of role associated with business unit
ur_role_execute	none	out	none	in	none	Visible permission; enabled if visible

Properties	Action					Description
	select	retrieve	insert	update	delete	
	7	4	7	4	7	
ur_read	none	out	none	in	none	Read permission; enabled if readable
ur_insert	none	out	none	in	none	Insert permission; enabled if insertable

Unit_User() Function

Provides access to the properties of the Web user access definition for a business unit.

The properties and actions for the Unit_User() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	7	4	7	4	7	
uu_unit_id	none	in	none	in	none	Unique business unit identifier
uu_user_id	none	out	none	in	none	Unique identifier of Web user associated with business unit
uu_user_name	none	out	none	none	none	Name of Web user associated with business unit
uu_execute	none	out	none	in	none	Visible permission; enabled if visible
uu_read	none	out	none	in	none	Read permission; enabled if readable
uu_insert	none	out	none	in	none	Insert permission; enabled if insertable

Unit_Location() Function

Provides access to the properties for a location associated with a business unit.

The properties and actions for the Unit_Location() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	7	4	7	4	
ul_unit_id	in	none	in	none	in	Unique business unit identifier
ul__location_id	out	none	in	none	none	Unique identifier for location associated with business unit
ul_location_name	out	none	none	none	none	Name of location associated with business unit

Unit_Copy() Function

Copies a business unit to a file to back it up or move it to another system.

The properties and actions for the Unit_Copy() function are described as follows:

Properties	Action		Description
	in	out	
	4	4	
unit_id	in	in	Unique business unit identifier
copy_file	in	none	Name of the file that contains the business unit documents and their descriptions
default_loc	in	none	Default location used when the original one does not match the new location

Document() Function

Provides access to the properties of a document.

The properties and actions for the Document() function are described as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	4	4	4	4	
doc_id	out	in	in	in	in	Unique document identifier
doc_type	out	out	in	none	none	Type of object: page or facet
doc_unit_id	out	out	in	in	none	Associated business unit identifier
doc_unit_name	out	out	in	none	none	Associated business unit name
doc_name	out	out	in	in	none	Document name
doc_suffix	out	out	in	in	none	Document extension
doc_public	out	out	in	in	none	Access flag tag; enabled if public
doc_pre_cache	out	out	in	in	none	Access flag tag; enabled if pre-cached
doc_perm_cache	out	out	in	in	none	Access flag tag; enabled if permanent
doc_session_cache	out	out	in	in	none	Access flag tag; enabled if session
doc_file	none	none	in	in	none	name of the remote user file
doc_ext_loc	out	out	in	in	none	Location identifier of the external file on the server
doc_ext_file	out	out	in	in	none	Name of the external file on the server
doc_ext_suffix	out	out	in	in	none	Extension of the external file on the server
doc_owner	out	out	none	none	none	Document owner
doc_transfer	none	none	none	in	none	Request to transfer an external document into the repository

Document_Role() Function

Provides access to the properties for a role associated with a document.

The properties and actions for the Document_Role() function are as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	7	4	7	4	7	
dr_doc_id	none	in	none	in	none	Unique document identifier
dr_role_id	none	out	none	in	none	Unique identifier for role associated with document
dr_role_name	none	out	none	none	none	Name of role associated with document
dr_role_execute	none	out	none	in	none	Visible permission; enabled if visible
dr_read	none	out	none	in	none	Read permission; enabled if readable
dr_insert	none	out	none	in	none	Insert permission; enabled if insertable

Document_User() Function

Provides access to the properties for a Web user associated with a document.

The properties and actions for the Document_User() function are as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	7	4	7	4	7	
du_doc_id	none	in	none	in	none	Unique document identifier
du_user_id	none	out	none	in	none	Unique identifier of user associated with Web user
du_user_name	none	out	none	none	none	Name of Web user associated with document
du_execute	none	out	none	in	none	Execute permission; enabled if document is executable
du_read	none	out	none	in	none	Read permission; enabled if document is readable

Properties	Action					Description
	select	retrieve	insert	update	delete	
	7	4	7	4	7	
du_update	none	out	none	in	none	Update permission; enabled if document can be updated
du_delete	none	out	none	in	none	Delete permission; enabled if document can be deleted

Session_Grp() Function

Provides access to the properties of a session group.

The properties and actions for the Session_Grp() function are as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	4	4	4	4	
sess_id	out	in	in	in	in	Unique session group identifier
sess_name	out	out	in	in	none	Session group name

Server Function

The function in this section allows you to access the server information in Web Deployment Option, including locations.

ICE_Locations() Function

Provides access to the properties of a location.

The properties and actions for the ICE_Locations() function are as follows:

Properties	Action					Description
	select	retrieve	insert	update	delete	
	4	4	4	4	4	
loc_id	out	in	in	in	in	Unique location identifier
loc_name	out	out	in	in	none	Location name
loc_path	out	out	in	in	none	File system directory specification for the location
loc_extensions	out	out	in	in	none	List of supported extensions
loc_http	out	out	in	in	none	Location type; enabled if HTTP visible
loc_ice	out	out	in	in	none	Location type; enabled if HTTP invisible
doc_public	out	out	in	in	none	Location type; enabled if public

Monitoring Functions

The functions in this section allow you to access information within Web Deployment Option that allows you to monitor activity. This includes active users, connected users, transactions, cursors, cached documents, and database connections.

Active_Users() Function

Provides access to the properties of an active user.

The properties and actions for the Active_Users() function are as follows:

Properties	Action		Description
	select	delete	
	4	4	
name	out	in	Unique active session identifier
ice_user	out	none	Remote user cookie and unique user session identifier
host	out	none	HTTP server or C client which issued the request
query	out	none	The SQL statement(s) issued by the active user

err_count	out	none	The number of errors that occurred during the active session

ICE_Users() Function

Provides access to the properties of a connected Web user.

The properties and actions for the ICE_Users() function are as follows:

Properties	Action		Description
	select	delete	
	4	4	
name	out	in	Unique user session identifier (cookie)
user	out	none	Web user name
req_count	out	none	Number of active users who are using this user session
timeout	out	none	The maximum idle time, in seconds, between requests to the server

ICE_User_Transactions() Function

Provides access to the properties of a user transaction.

The properties and actions for the ICE_User_Transactions() function are as follows:

Properties	Action		Description
	select	delete	
	4	4	
key	out	in	Unique user identifier
name	out	none	Transaction name
owner	out	none	Transaction owner
connection	out	none	Identifier of the database connection for the transaction

ICE_User_Cursors() Function

Provides access to the properties of a user cursor.

The properties and actions for the ICE_User_Cursors() function are as follows:

Properties	Action		Description
	select	delete	
	4	4	
key	out	in	Unique user identifier
name	out	none	Cursor name
owner	out	none	Cursor owner
query	out	none	SQL statement(s) that initiated the cursor

ICE_Cache() Function

Provides access to the properties of a cached file (page or facet).

The properties and actions for the ICE_Cache() function are as follows:

Properties	Action		Description
	select	delete	
	4	4	
key	out	in	Unique file identifier
name	out	none	The name of the file in the cache
loc_name	out	none	The name of the location in which the cached file resides
status	out	none	Indicates whether the file is usable or unusable
exist	out	none	Indicates that the file is from the file system, rather than the repository. The value is 0 if the file is stored in the database or is a temporary file.
file_counter	out	none	The number of times the file has been requested currently
owner	out	none	User session that is using this file

Properties	Action		Description
	select	delete	
	4	4	
timeout	out	none	The amount of time (in seconds) before the file is removed from the cache
in_use	out	none	Indicates whether the session is using this file. If in use, the file cannot be deleted or refreshed.
req_count	out	none	The number of requests made to the file by the session identified by the requestor (cookie)

ICE_Connect_Info() Function

Provides access to the properties of a database connection.

The properties and actions for the ICE_Connect_Info() function are as follows:

Properties	Action		Description
	select	delete	
	4	4	
key	out	in	Unique identifier for the database connection
driver	out	none	The driver that Web Deployment Option is using to communicate with the data source
dbname	out	none	Database or database connection name
used	out	none	Indicates whether the database connection is currently active (value = 1).
timeout	out	none	The amount of time, in seconds, before the database connection will be closed

Additional Functions

The functions in this section provide additional functionality in the Web Deployment Option environment.

TagToString() Function

Replaces reserved HTML characters by their string equivalents.

The properties and actions for the TagToString() function are as follows:

Properties	Action	Description
tag	in	String with tags
string	out	Converted string

Dir() Function

Lists the files in a specified location.

The properties and actions for the Dir() function are as follows:

Properties	Action	Description
location_id	in	Location identifier
prefix	in/out	File name
suffix	out	File extension

GetVariables() Function

Obtains the properties for the variables that are currently declared for the specified scope.

The properties and actions for the GetVariables() function are as follows:

Properties	Action	Description
page	in/out	"checked" if the variable is a page variable
session	in/out	"checked" if the variable is a session variable
server	in/out	"checked" if the variable is a server variable
name	out	Variable name
value	out	Variable value

Appendix E: Using an XML Authoring Tool

With an XML content enabling application, such as SoftQuad XMetaL, you can add Web Deployment Option XML elements to your documents with ease. The Web Deployment Option XML elements from the DTD are accessible through these types of tools, allowing you to construct your Web applications using the visual interface of your choice.

In this chapter, the SoftQuad XMetaL 2.0 tool is used to demonstrate the use of Web Deployment Option tags in this type of development environment.

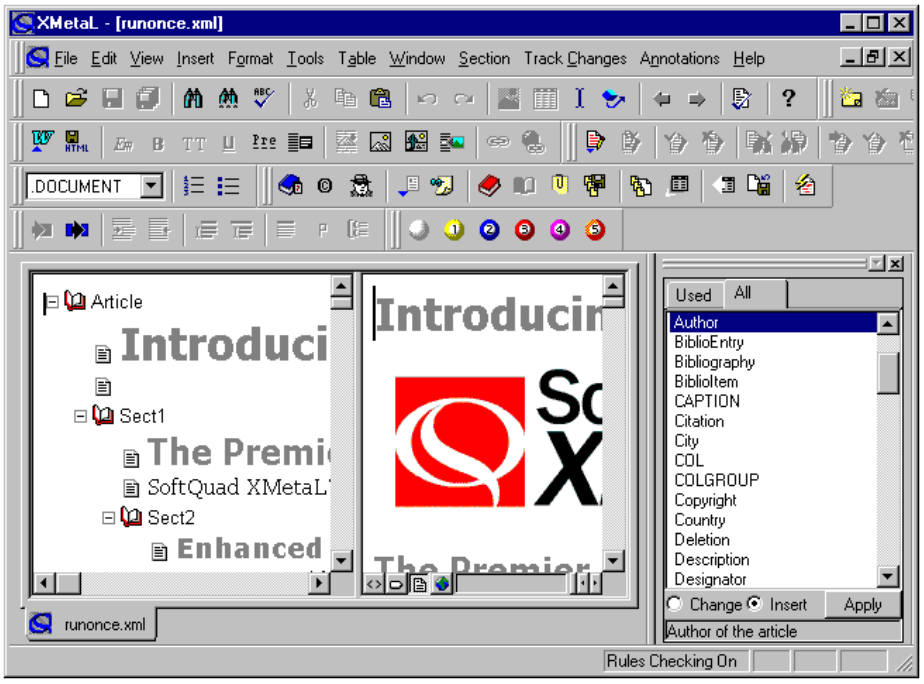
Note: To use a particular tool with Web Deployment Option extensions, it must support the xhtml1-transitional DTD that is published by the W3C.

Starting XMetaL

Windows

To open the XMetaL application and begin working, navigate to and select the XMetaL program through the Start menu in Windows.

The application window appears with the last saved workspace configuration. For example:



In this workspace, the Structure view pane appears to the left of the Document window. Also shown is the Element List window, which displays the various elements (based on the current DTD) that can be added to the document.

The next step involves creating a new application.

Creating a New Document

The first steps you will want to perform are to create a new document and attach the Web Deployment Option DTD. To do this:

1. Choose File, New.

   The New dialog appears.

2. Select the Blank XML Document icon and click OK.

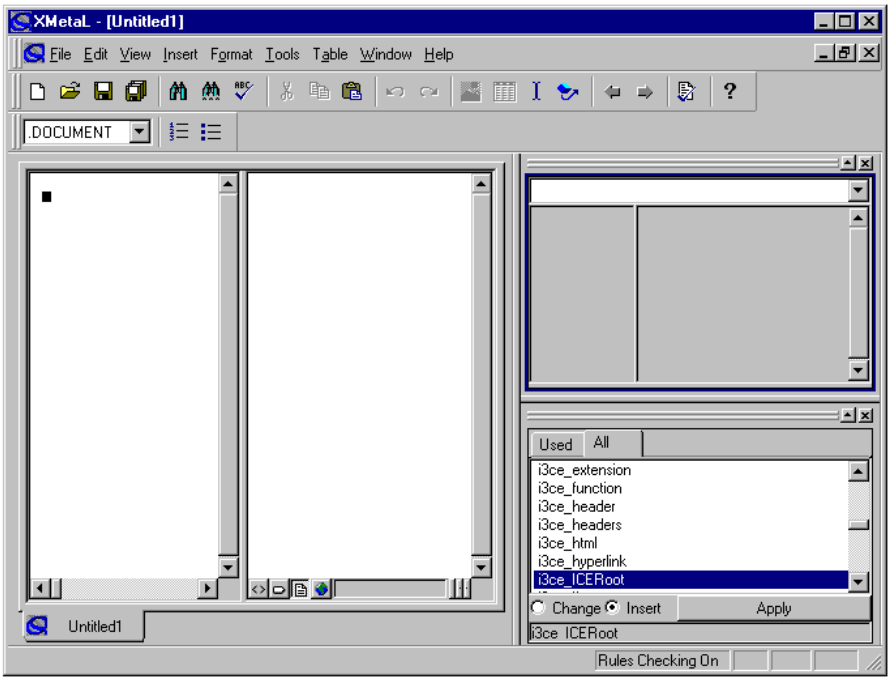
   The Choose a DTD or Rules File dialog appears.

3. Locate and select the xhtml1-transitional.dtd file, which resides in the following path:

   Ingres system drive and directory\ingres\ice\DTD

4. Click Open.

   The workspace appears similar to the following, with the Web Deployment Option macro tags displayed in the Element List:



> **Tip:** In the Element List window, all the Web Deployment Option elements are prefixed by "i3ce_", causing them to appear together, making them easier to find.

   Note: Your workspace may appear different than this one, depending on how it was configured when you last saved a document.

5. We are going to change our workspace a little, adding the Attribute Inspector window, disabling the Structure view, and enabling the Tags On view in the Document window. Use the View menu commands to accomplish this.

6. Using the File, Save command, save the document using the name my_query.

Building Macro Elements

Next, we will use the XMetaL environment to reproduce one of the query elements shown in the example in the i3ce_query tag section in "Chapter 5: Using the Macro Language."

The example, shown below, selects all columns from the plays table in the iceTutor database. It creates a transaction and a cursor. Then, it retrieves five rows at a time and formats them into an XHTML table.

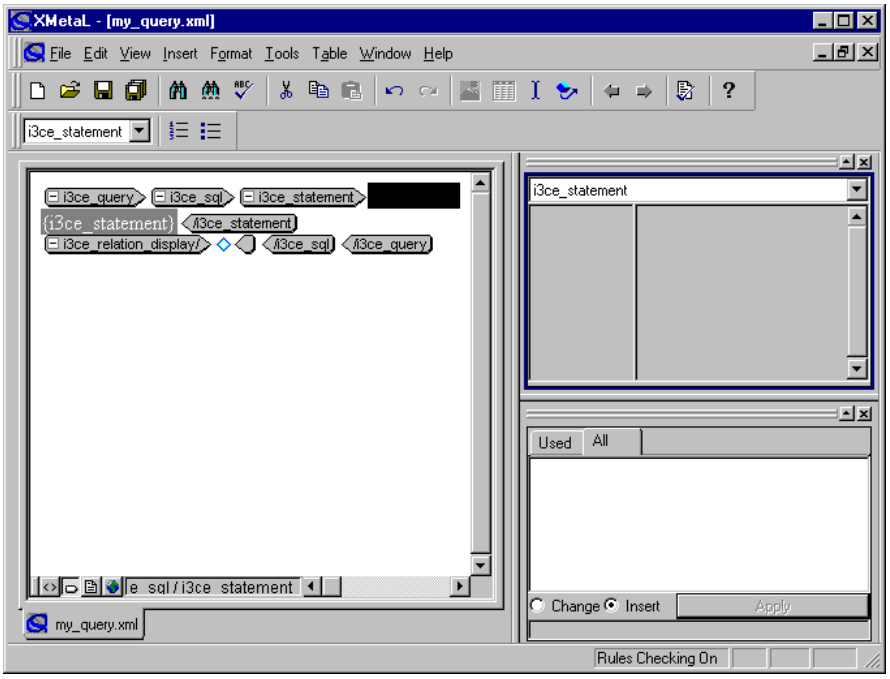
```
<i3ce_query i3ce_database="icetutor">
    <i3ce_sql i3ce_transaction="myTransaction" i3ce_cursor="myCursor">
        <i3ce_statement>
            select * from plays
        </i3ce_statement>
        <i3ce_rowsPerRequest i3ce_rowcount="5"/>
        <i3ce_relation_display i3ce_typename="i3ce_table"/>
    </i3ce_sql>
</i3ce_query>
```

Now, let's move on to inserting this macro tag into the XMetaL document.

Using the XMetaL Environment

Adding the <i3ce_query> Element

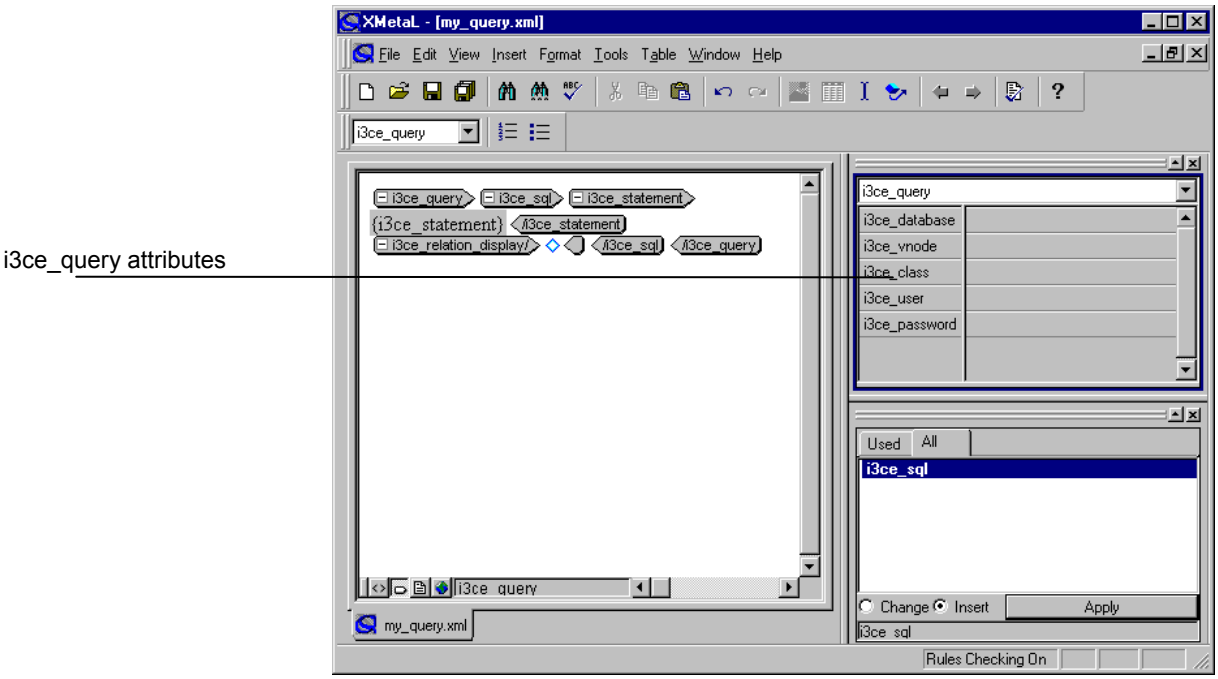
In the Element List window, select the <i3ce_query> element and, with the Insert option selected, click Apply. The element is inserted into the Document window as follows:



Defining <i3ce_query> Attributes

You will notice that in Tags On view, you can see a graphical view of the tags and child tags associated with i3ce_query.

Before we get to the SQL statement, let us define the attributes associated with the <i3ce_query> element. To do this, place the insertion point directly after the <i3ce_query> element, which displays the definable attributes for this element in the Attribute Inspector:

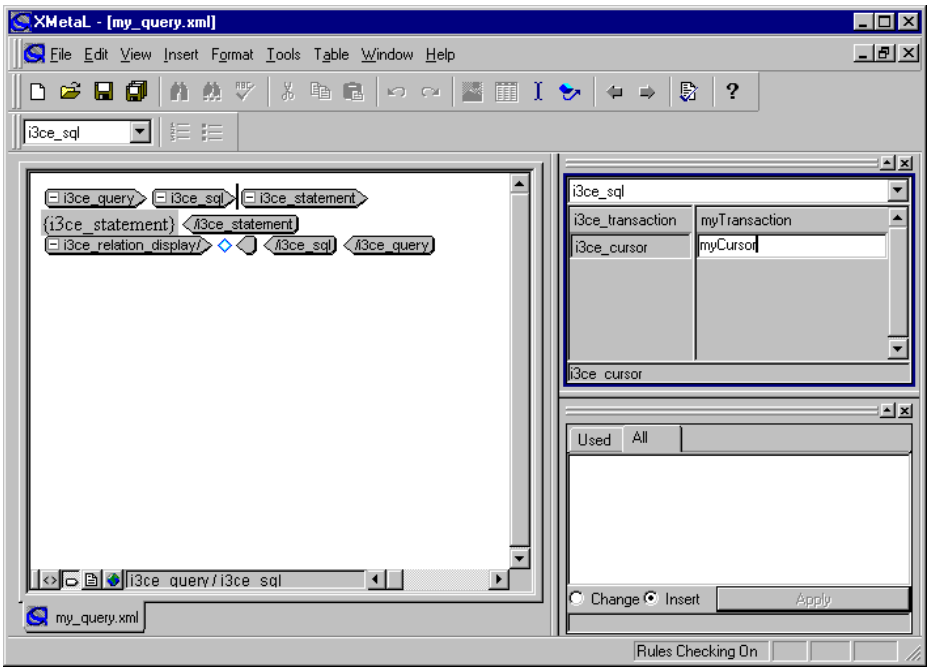


Now, let's enter the value for the i3ce_database attribute, as defined in the example. Click in the area next to the i3ce_database attribute. Type **iceTutor** and press Enter (or click outside the field).

Note: You do not have to include the quotation marks around the attribute values, as this is done automatically for you.

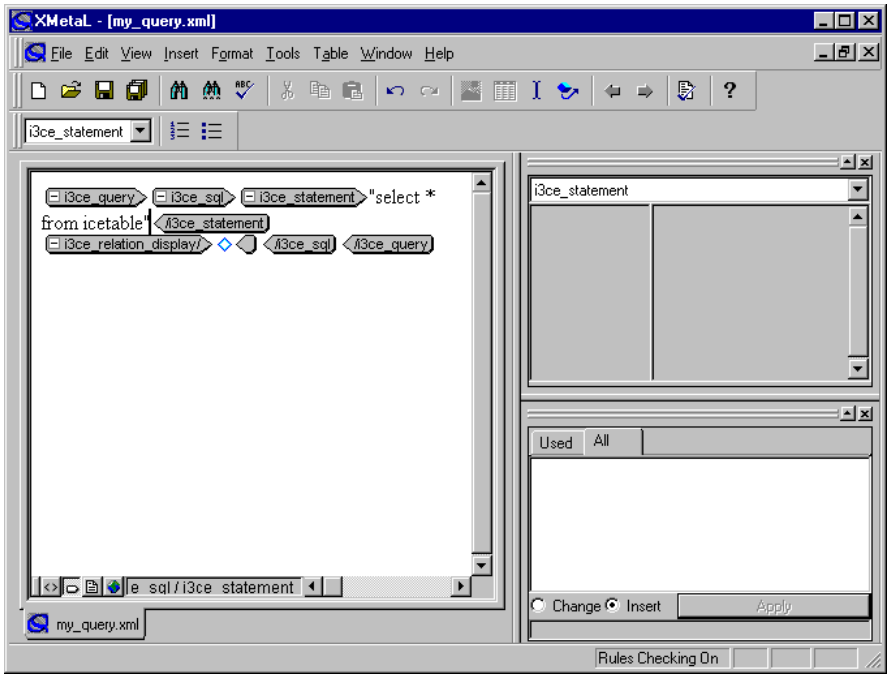
Defining <i3ce_sql> Attributes

Now, we'll define the attributes for the child element, <i3ce_sql>. Again, place the insertion point in the <i3ce_sql> element, displaying the i3ce_transaction and i3ce_cursor attributes. Enter the values **myTransaction** and **myCursor** for these attributes:



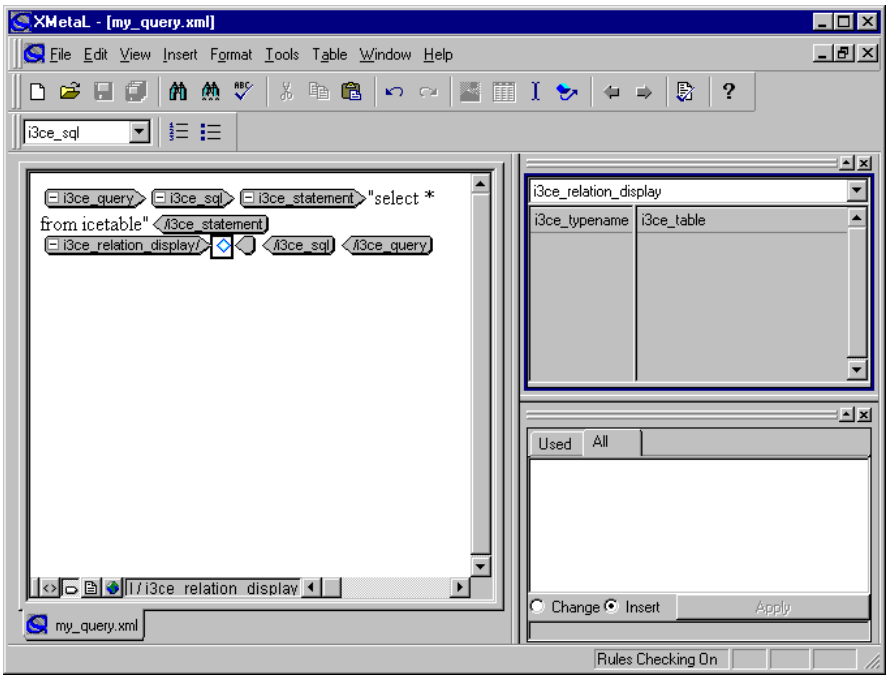
Defining the <i3ce_sql> Statement

Within the <i3ce_sql> element, you will see the <i3ce_statement> element. Select this element and enter the statement **select * from plays** directly into the Document window pane where the {i3ce_statement} placeholder appears, as follows:

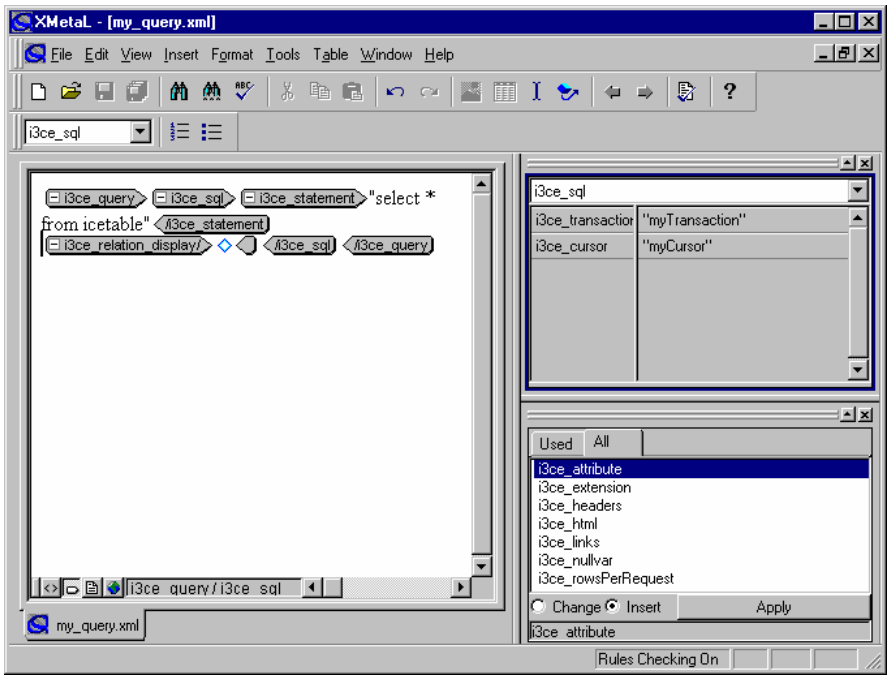


Adding <i3ce_sql> Child Tags

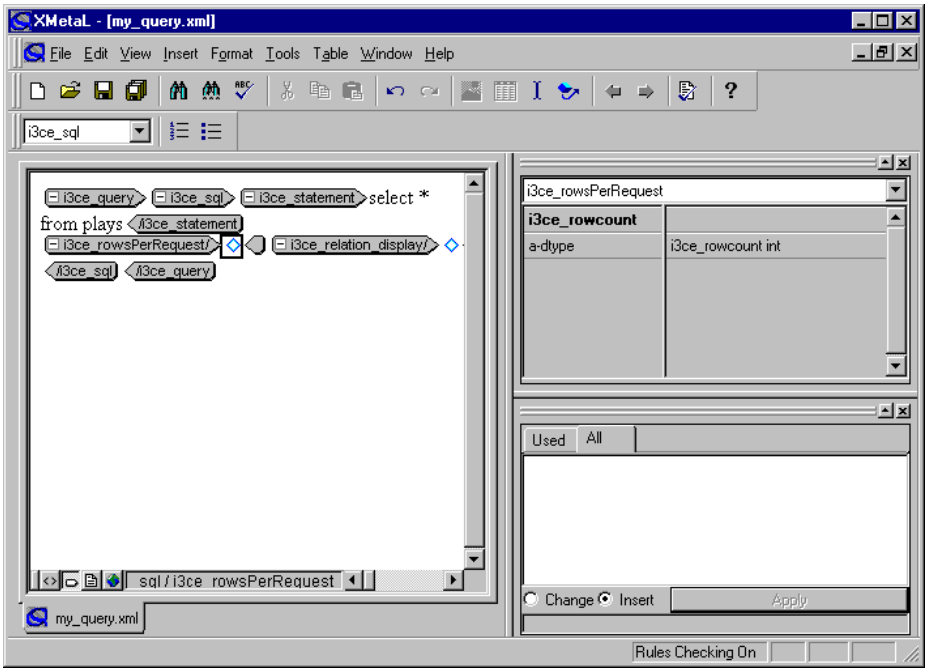
The next elements that need to be added are the <i3ce_rowsPerRequest> and <i3ce_relation_display> elements. In Tags On view, click on the diamond icon after <i3ce_relation_display> and you will see that the i3ce_typename attribute is already set to i3ce_table, which is the value we need (if necessary, you could have clicked in this edit control, displaying a drop-down list of other valid values).



Clicking before the <i3ce_relation_display> element, notice the Element List displays a list of valid elements at this point in the <i3ce_query> element.



In the Element List window, select the <i3ce_rowsPerRequest> element and, while Insert is selected, click Apply. The <i3ce_rowsPerRequest> element is added to the Document window. In the Attribute Inspector window, you will notice that attribute i3ce_rowcount appears:



In the i3ce_rowcount text box, enter "5" and press Enter (or click outside the field). This completes the creation of the first <i3ce_query> element in the example. You can save the document by choosing File, Save.

Translating the XML File

You are now ready to translate the XML file into an ICE Template file. To do this, you must use the ICETranslate utility, whose syntax is shown below. Note that the HTML file must also be registered with the Web server.

*input_xml_file.*xml **>** *output_ICE_HTML_macro_file*.html

Type the following at the command prompt (assuming you saved the file as "my_query.xml"):

ICETranslate my_query.xml > my_query.html

Note: On UNIX, the first four letters of this command must be uppercase. On Windows, it is not necessary.

The my_query.html file should contain the following:

```
<!-- #ICE
    DATABASE=`icetutor`
    TRANSACTION=`my_Transaction`
    CURSOR=`my_Cursor`
    SQL=`select * from plays`
    ROWS=`5`
    TYPE=`TABLE`
-->
```

Note: The error checker in XMetaL detects if there are any problems in your document and notifies you so that you can make debug the program on the fly.

So you see how quick and easy the process is to build XML programs and translate them into HTML macro template files! You can use the same procedure to construct other XML elements that will make up your XML document.

Index

A

accessing Web Deployment Option pages, B-7

Active_Users() function, D-12

Apache Web Server, 2-11

applications
- creating, 6-1
- example of a data browsing application, 6-28
- example of an Internet shopping application, 6-54

associating
- business units with Web users, 4-4
- database connections with profiles, 6-27
- database connections with Web users, 4-4
- facets with business units, 6-22
- locations with business units, 4-18, 6-21
- pages and facets with Web users, 4-4
- pages with business units, 6-21
- roles with business units, 6-27
- roles with profiles, 6-26
- roles with Web users, 4-4

B

business unit functions
- Document() function, D-8
- Document_Role() function, D-9
- Document_User() function, D-10
- Session_Grp() function, D-11
- Unit() function, D-6
- Unit_Copy() function, D-8
- Unit_Location() function, D-7
- Unit_Role() function, D-6
- Unit_User() function, D-7

business units
- associating with facets, 6-22
- associating with locations, 6-21
- associating with pages, 6-21
- associating with roles, 6-27
- creating, 6-20
- defined, 4-13
- managing, 4-13
- registering and deregistering multiple files, 4-14
- use of, 4-14

C

C API
- ICE_C_CLIENT structure, 7-9
- ICE_C_Close() function, 7-1
- ICE_C_Connect() function, 7-2
- ICE_C_Disconnect() function, 7-3
- ICE_C_Execute() function, 7-3
- ICE_C_Fetch() function, 7-5
- ICE_C_GetAttribute() function, 7-6
- ICE_C_Initialize() function, 7-7
- ICE_C_LastError() function, 7-8
- ICE_C_PARAMS structure, 7-9
- ICE_STATUS data type, 7-8
- using, 7-1

C API, sample, 7-11

COMMIT keyword, 5-22

Configuration Manager, 2-15

configuring
- ICE Server, 2-14
- IIS, 2-5
- the Apache Web Server, 2-11
- the HTTP server, 2-1

creating
- applications, 6-1
- business units, 6-20
- database connections, 6-24
- locations, 6-18
- profiles, 6-24
- roles, 6-25
- session groups, 6-18

D

data sources, granting access to, 4-3

database connections
- associating with profiles, 6-27
- creating, 6-24
- defined, 4-6
- managing, 4-6
- usage of, 4-7

database users
- defined, 4-5
- managing, 4-5
- usage of, 4-6

Database() function, D-2

DBUser() function, D-1

DECLARE keyword, 5-23

designing applications pages, 6-1

dialogs
- Associate a Location to Business Unit, 6-21
- Associate DB Connection to ICE Profile, 6-27
- Associate Role to ICE Profile, 6-26
- Create ICE Business Unit, 6-20
- Create ICE Database Connection, 6-24
- Create ICE Facet for Business Unit, 6-23
- Create ICE Location, 6-19
- Create ICE Page for Business Unit, 6-22
- Create ICE Profile, 6-25
- Create ICE Role, 6-26
- Create ICE Session Group, 6-18
- Enter Login/Password for Accessing ICE Information, 4-2
- Role Access Definition for Business Unit, 6-28

Dir() function, D-16

Document() function, D-8

Document_Role() function, D-9

Document_User() function, D-10

documents, defined, 4-15

E

extension functions
- calling from a Web page, 8-4
- description, 8-2
- initialization, 8-1
- sample, 8-5
- syntax description, 8-3

F

facets
- associating with business units, 6-22
- defined, 4-15
- managing, 4-15
- usage of, 4-16

FUNCTION keyword, 5-25

G

GetVariables() function, D-16

granting access to Web resources, 4-3

H

HTML
- basics, B-1
- documents, B-1
- elements generated by Web Deployment Option, B-7
- elements used by Web Deployment Option, B-5

HTTP server. *See* web server

I

ICE server
- addressing, 6-2

ICE Server

configuring, 2-14
extension functions, 8-1
ICE_C_CLIENT structure, 7-9
ICE_C_Close() function, 7-1
ICE_C_Connect() function, 7-2
ICE_C_Disconnect() function, 7-3
ICE_C_Execute() function, 7-3
ICE_C_Fetch() function, 7-5
ICE_C_GetAttribute() function, 7-6
ICE_C_Initialize() function, 7-7
ICE_C_LastError() function, 7-8
ICE_C_PARAMS structure, 7-9
ICE_Cache() function, D-14
ICE_Connect_Info() function, D-15
ICE_Locations() function, D-11
ICE_STATUS data type, 7-8
ICE_User_Cursors() function, D-14
ICE_User_Transactions() function, D-13
ICE_Users() function, D-13
IF keyword, 5-27
IIS, 2-5
INCLUDE keyword, 5-28
installing web server, 2-1

K

keywords, used with Web Deployment Option macros, 5-22

L

locations
  associating with business units, 6-21
  creating, 6-18, 6-19
  defined, 4-10
  managing, 4-11
  usage of, 4-11
  use for, 4-14
  use of, 4-10
logging into the Web Deployment Option, 4-2

M

macro language
  keywords, 5-22
  macro tag syntax, 5-2
  macro tags, 5-2
  statement syntax, 5-21
  using macros, 5-21
macro language keywords, 5-22
  COMMIT, 5-22
  DECLARE, 5-23
  FUNCTION, 5-25
  IF, 5-27
  INCLUDE, 5-28
  ROLLBACK, 5-30
  SQL, 5-31
  SWITCH, 5-39
  VAR, 5-41
managing
  business units, 4-13
  database connections, 4-6
  database users, 4-5
  locations, 4-11
  pages and facets, 4-15
  profiles, 4-8
  role access definitions, 4-16
  roles, 4-7
  server variables, 4-12
  session groups, 4-9
  Web Deployment Option objects, 4-1, 6-18
  Web user access definitions, 4-17
  Web users, 4-3
Microsoft Internet Information Server, 2-5
monitoring functions
  Active_Users() function, D-12
  ICE_Cache() function, D-14
  ICE_Connect_Info() function, D-15
  ICE_User_Cursors() function, D-14
  ICE_User_Transactions() function, D-13

ICE_Users() function, D-13

monitoring Web Deployment Option information, 4-19

P

pages
associating with business units, 6-21
creating for applications, 6-1
defined, 4-15
managing, 4-15
usage of, 4-16

Performance Monitor, viewing Web Deployment Option information, 4-19

permissions, for pages and facets, 4-18

Plays tutorial application
data used, 6-73
re-creating, 6-2
touring, 6-2

Profile() function, D-4

Profile_Database() function, D-5

Profile_Role() function, D-5

profiles
associating with database connections, 6-27
associating with roles, 6-26
creating, 6-24
defined, 4-8
managing, 4-8
use of, 4-9

public files, setting up, 6-19

R

reserved words in Web Deployment Option, C-1

role access definitions
defined, 4-16
managing, 4-16
usage of, 4-17

Role() function, D-2

roles
associating with business units, 6-27
associating with profiles, 6-26
creating, 6-25
defined, 4-7
managing, 4-7
usage of, 4-8

ROLLBACK keyword, 5-30

S

security
managing, 4-3
setting up, 6-1

security functions
Database() function, D-2
DBUser() function, D-1
Profile() function, D-4
Profile_Database() function, D-5
Profile_Role() function, D-5
Role() function, D-2
User() function, D-2
User_Database() function, D-4
User_Role() function, D-3

server extension functions
ICE_Locations() function, D-11

server functions, miscellaneous
Dir() function, D-16
GetVariables() function, D-16
TagToString() function, D-16

server variables
defined, 4-11
managing, 4-12
usage of, 4-12

session groups
creating, 6-18
defined, 4-9
managing, 4-9
usage of, 4-10

Session_Grp() function, D-11

SQL keyword, 5-31

SWITCH keyword, 5-39

syntax
for macro statements, 5-21
for macro tags, 5-2

T

TagToString() function, D-16

tutorial application, 6-1

U

Unit() function, D-6

Unit_Copy() function, D-8

Unit_Location() function, D-7

Unit_Role() function, D-6

Unit_User() function, D-7

User() function, D-2

User_Database() function, D-4

User_Role() function, D-3

V

VAR keyword, 5-41

variables, using, 4-12

Visual DBA, using with Web Deployment Option, 4-1

W

Web Deployment Option
architecture, 3-3
business unit functions. *See* business unit functions
C API. *See* C API
description of users, 3-2
logging into, 4-2
macro language. *See* macro language
miscellaneous functions. *See* server functions, miscellaneous
monitoring functions. *See* monitoring functions
overview, 3-1
reserved words, C-1
security functions. *See* security functions
server extension functions. *See* server extension functions
viewing and managing information, 4-1
web site components, 3-4
XML tag set, using, 5-1

Web resources, managing, 4-3

web server
configuring, 2-1
Document Root Directory, 2-15
installing, 2-1
supported, 2-1

Web user access definitions
defined, 4-17
managing, 4-17
usage of, 4-18

Web users
defined, 4-3
managing, 4-3
usage of, 4-4

X

XML macro tags, 5-1, 5-2