Oracle® HTML DB

User's Guide

Release 1.5

Part No. B10992-01

December 2003

ORACLE®

Oracle HTML DB User's Guide, Release 1.5

Part No. B10992-01

Primary Author: Terri Winters

Contributor: Christina Cho, Michael Hichwa, Joel Kallman, Sergio Leunissen, Raj Mattamal, Tyler Muth, Marc Sewtz, Scott Spadafore, and Jason Straub

Contents

3 Running a Demonstration Application

Part II Using Oracle HTML DB

4 Managing Data with Data Workshop

5 Using SQL Workshop to Manage Database Objects

6 Application Builder Concepts

7 Using Application Builder

8 Building Application Components

9 Debugging an Application

10 Managing an Application

11 Managing Your Development Workspace

12 Advanced Programming Techniques

13 Oracle HTML DB APIs

15 Managing Services

16 Managing Globalization

A Available Conditions

Index

Send Us Your Comments

Oracle HTML DB User's Guide, Release 1.5

Part No. B10992-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  U.S.A.

If you would like a reply, please give your name, address, telephone number, and (optionally) your electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Oracle HTML DB User's Guide describes how to use the Oracle HTML DB development environment to build and deploy database-centric Web applications. Oracle HTML DB turns a single Oracle database into a shared service by enabling multiple workgroups to build and access applications as if they were running in separate databases.

This preface contains these topics:

Intended Audience

Oracle HTML DB User's Guide is intended for application developers who are building database-centric Web applications using Oracle HTML DB. The guide describes how to use the Oracle HTML DB development environment to build, debug, manage, and deploy applications. To use this guide, you need to have a general understanding of relational database concepts as well as an understanding of the operating system environment under which you are running Oracle HTML DB.

Organization

This document contains:

Part I, "Getting Started with Oracle HTML DB"

Part I provides an introduction to Oracle HTML DB by introducing you to basic Oracle HTML DB concepts.

Chapter 1, "What is Oracle HTML DB?"

This chapter offers a general description of Oracle HTML DB and the components you can use it to develop database-centric Web applications.

Chapter 2, "Quick Start"

This chapter offers a quick introduction to using Oracle HTML DB.

Chapter 3, "Running a Demonstration Application"

This chapter describes how to run a demonstration application and defines fundamental concepts that are unique to Oracle HTML DB.

Part II, "Using Oracle HTML DB"

Part II describes how to use Data Workshop, SQL Workshop, and Application Builder to develop database-driven applications.

Chapter 4, "Managing Data with Data Workshop"

This chapter describes how to use Data Workshop to import data into and export data from a hosted database.

Chapter 5, "Using SQL Workshop to Manage Database Objects"

This chapter provides information on how to use SQL Workshop to view and manage database objects as well as browse the data dictionary.

Chapter 6, "Application Builder Concepts"

This chapter provides basic conceptual information about Application Builder.

Chapter 7, "Using Application Builder"

This chapter describes how to use Application Builder to build the pages that comprise an application.

Chapter 8, "Building Application Components"

This chapter describes how to build application components in Oracle HTML DB, including navigation, regions, buttons, Lists of Values, forms, reports, charts, and help pages.

Chapter 9, "Debugging an Application"

This chapter describes a number of approaches to debugging your application including viewing Debug Mode, enabling SQL tracing, viewing page reports, and how to manually remove a component to isolate a problem.

Chapter 10, "Managing an Application"

This chapter provides information about Application Builder utilities, how to export and import an application, and how to manage application security.

Chapter 11, "Managing Your Development Workspace"

This chapter describes the tools and reports available to Workspace administrators.

Chapter 12, "Advanced Programming Techniques"

This chapter provides information about advanced programming techniques including establishing database links, using collections, running background SQL, utilizing Web Services and managing user preferences.

Chapter 13, "Oracle HTML DB APIs"

This chapter describes available Oracle HTML DB APIs.

Part III, "Administration"

Part III describes the tasks associated with administering Oracle HTML DB, including creating and managing workspaces, translating an application, and managing activities, log files, and sessions.

Chapter 14, "Administering Workspaces"

This chapter describes tasks an Oracle HTML DB administrator performs when administering workspaces.

Chapter 15, "Managing Services"

This chapter provides information about additional administrator activities available in Oracle HTML DB, including sending e-mail, monitoring user activity, managing log files, and managing sessions.

Chapter 16, "Managing Globalization"

This chapter describes how to translate an application created in Oracle HTML DB.

Appendix A, "Available Conditions"

Provides a listing of conditions available in Oracle HTML DB.

Related Documentation

For more information, see these Oracle resources:

- ***Oracle Database Concepts***
- ***Oracle Database Application Developer's Guide - Fundamentals***
- ***Oracle Database Administrator's Guide***
- ***Oracle Database SQL Reference***

Many of the examples in this book use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/membership/`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/documentation/`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
`[ ]`	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (digits [ , precision ])`
`{ }`	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
`|`	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`

`...`	Horizontal ellipsis points indicate either:
	■ That we have omitted parts of the code that are not directly related to the example
	■ That you can repeat a portion of the code
		`CREATE TABLE ... AS subquery;`
		
		`SELECT col1, col2, ... , coln FROM employees;`

. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	`SQL> SELECT NAME FROM V$DATAFILE;`
		`NAME`
		`----------`
		`/fs1/dbs/tbs_01.dbf`
		`/fs1/dbs/tbs_02.dbf`
		`.`
		`.`
		`.`
		`/fs1/dbs/tbs_09.dbf`
		`9 rows selected.`

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`

Convention	Meaning	Example
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Part I

Getting Started with Oracle HTML DB

Part I provides an introduction to Oracle HTML DB. These chapters introduce you to basic Oracle HTML DB concepts.

Part I contains the following chapters:

1

What is Oracle HTML DB?

The section offers a general description of Oracle HTML DB and the components you can use to develop database-centric Web applications.

This section contains the following topics:

About Oracle HTML DB

Oracle HTML DB is a hosted declarative development environment for developing and deploying database-centric Web applications. Oracle HTML DB turns a single Oracle database into a shared service by enabling multiple workgroups to build and access applications as if they were running in separate databases. Thanks to built-in features such as design themes, navigational controls, form handlers, and flexible reports, Oracle HTML DB accelerates the application development process.

The HTML DB engine renders applications in real time from data stored in database tables. When you create or extend your application, Oracle HTML DB creates or modifies metadata stored in database tables. When the application is run, the HTML DB engine then reads the metadata and displays the application.

Oracle HTML DB automatically maintains session state without requiring any coding. To provide stateful behavior within an application Oracle HTML DB transparently manages session state in the database. Application developers can get and set session state using simple substitutions as well as standard SQL bind variable syntax.

The Oracle HTML DB development platform consists of the following components:

- Application Builder
- SQL Workshop
- Data Workshop

About Application Builder

You use Application Builder to assemble an HTML interface (or application) on top of database objects such as tables and procedures. An application is a collection of database-driven Web pages linked together using tabs, buttons, or hypertext links. Once you create an application, the HTML DB engine renders the application using templates and UI elements you specify.

A page is the basic building block of an application. Each page can have buttons and fields and can include application logic (or processes). You can branch from one page to the next using conditional navigation, perform calculations, run validations (such as edit checks), and display reports, forms, and charts.

See Also:

- "Application Builder Concepts" on page 6-1
- "Using Application Builder" on page 7-1
- "Building Application Components" on page 8-1

About SQL Workshop

You use SQL Workshop to view and manage database objects from a Web browser. Using SQL Workshop you can store and retrieve data, execute SQL commands, and perform the following tasks:

- Run SQL commands
- Upload and run SQL scripts
- Maintain a history of the executed SQL
- Create or modify database objects
- Query data by example
- Browse the data dictionary
- Enable database browsing with drill-up and drill-down

See Also: "Using SQL Workshop to Manage Database Objects" on page 5-1

About Data Workshop

You use Data Workshop to import data into and export data from the hosted database. Supported import formats include text (such as comma or tab delimited data), XML documents, and spreadsheets. Supported export formats include text (such as comma or tab delimited data) and XML documents.

For example, you can quickly share data with multiple users by converting a spreadsheet into a database table using the Spreadsheet Import Wizard. Running this wizard creates a new table and loads the data without requiring any SQL knowledge. Once the data is loaded into a database table, you can build a application on top of it just like you would on any other database table.

See Also:

- "Managing Data with Data Workshop" on page 4-1
- "Importing Spreadsheet Data" on page 4-3

2

Quick Start

This section offers a quick introduction to using Oracle HTML DB. This section assumes you have already completed the installation process.

This section contains the following topics:

- Understanding Oracle HTML DB User Roles
- Logging in to Oracle HTML DB
- About Oracle HTML DB User Interface
- Creating an Application Using the Create Application Wizard

See Also:

- "Running a Demonstration Application" on page 3-1
- "Application Builder Concepts" on page 6-1
- "Using Application Builder" on page 7-1

Understanding Oracle HTML DB User Roles

In the Oracle HTML DB development environment, users log in to a shared work area called a workspace. Users are divided into three primary roles:

- Developer
- Workspace administrator
- Oracle HTML DB administrator

A developer can create and edit applications. A Workspace administrator performs administrator tasks specific to their workspace. An Oracle HTML DB administrator manages an entire Oracle HTML DB development environment instance.

See Also:

- "Application Builder Concepts" on page 6-1
- "Managing Your Development Workspace" on page 11-1
- "Administering Workspaces" on page 14-1

Logging in to Oracle HTML DB

When you log in to Oracle HTML DB, you log in to a workspace. A workspace is an area within the Oracle HTML DB development environment where multiple developers can create applications.

Topics in this section include:

- Requesting a Workspace
- Logging in to a Workspace
- Resetting Your Password
- Logging Out of Your Workspace

Note: Before users can request a workspace or change their passwords, an Oracle HTML DB administrator must configure the engine settings.

See Also: "Managing Engine Settings" on page 15-5

Requesting a Workspace

Note: This section only applies if your Oracle HTML DB administrator has configured Oracle HTML DB to support workspace requests.

See Also: "Specifying a Provisioning Mode" on page 14-3 for more information on enabling workspace requests

Before you can log in Oracle HTML DB, you must request a workspace. Each workspace has a unique ID and name. Only an administrator with the appropriate credentials can create a new workspace.

To request a workspace:

1. In a Web browser, navigate to the Oracle HTML DB Login page. By default, Oracle HTML DB installs to the following location:

   ```
   http://server:port/pls/Database Authentication Descriptor/htmldb
   ```

   The Login page appears.

2. Under Tasks, click **Request a Workspace**.

   The Request Service Wizard appears.

3. Click **Continue** and follow the on-screen instructions.

See Also: "Creating a Workspace" on page 14-3

Logging in to a Workspace

Once your workspace request has been approved, an Oracle HTML DB administrator provides you with a workspace name, username, and password.

To log in to Oracle HTML DB:

1. In a Web browser, navigate to the Oracle HTML DB Login page. By default, Oracle HTML DB installs to the following location:

   ```
   http://server:port/pls/Database Authentication Descriptor/htmldb
   ```

   The Login page appears.

2. Under Login, type the following:
   - In Workspace, type the name of your workspace.
   - In Username, type a username.
   - In Password, type a case sensitive password.

3. Click **Login**.

Resetting Your Password

You can reset your password by clicking the Reset Password link on the Oracle HTML DB Login page.

To reset your password:

1. In a Web browser, navigate to the Oracle HTML DB Login page. By default, Oracle HTML DB installs to the following location:

   ```
   http://server:port/pls/Database Authentication Descriptor/htmldb
   ```

2. Under Tasks, click **Reset Password**.
3. Type you workspace name and e-mail address and click **Reset Password**. A new password is sent to your e-mail address.

Logging Out of Your Workspace

To logout of Oracle HTML DB, click the **Logout** icon in the upper right corner of the window.

About Oracle HTML DB User Interface

Once you log in to Oracle HTML DB, the Oracle HTML DB Home page appears as shown in Figure 2–1.

Figure 2–1 Oracle HTML DB Home Page

The Oracle HTML DB development environment consists of three components:

- **Application Builder**. Use Application Builder to assemble an HTML interface (or application) on top of a database objects such as tables and procedures.
- **SQL Workshop**. Use SQL Workshop to view and manage database objects from a Web browser.
- **Data Workshop**. Use Data Workshop to import data into and export from the hosted database.

To access any of these components, click the large icons in the center of the page, or click the navigation bar icons in upper right corner. Use Administration to access the Administration Services page.

See Also:

- "Application Builder Concepts" on page 6-1
- "Using Application Builder" on page 7-1
- "Using SQL Workshop to Manage Database Objects" on page 5-1
- "Managing Data with Data Workshop" on page 4-1
- "Managing Your Development Workspace" on page 11-1

About Using the Tasks List

Many Oracle HTML DB pages feature a Tasks list on the right side of the page. Select a list item to quickly link to common procedures.

Other Sources of Information

Most pages in Oracle HTML DB include page level help. Page level help displays in a text box on the right side of the page and offers a brief description of the page functionality. Oracle HTML DB also includes two other forms of online help:

- **Procedural online help.** You can access an HTML-based online help system by clicking the Help navigation bar icon.
- **Field level help.** Most lists of values, select lists, check boxes, and fields in Oracle HTML DB include item help. When item help is available, the item label appears highlighted when you pass your cursor over it. Clicking the item label displays a a description in a separate window.

Creating an Application Using the Create Application Wizard

A quick way to make data in the Oracle database accessible to an end user is to run the Create Application Wizard. This wizard creates a basic application which contains up to five pages and includes:

- a home page with a menu
- a searchable report
- an edit page
- an insert page
- a charting page

The Create Application Wizard assumes you have a single table or several unrelated tables for which you want to create a report or update and insert data. Once the application is generated, you can modify it using Application Builder.

To create an application using the Create Application Wizard:

1. Log in to Oracle HTML DB as described in "Logging in to Oracle HTML DB" on page 2-2.
2. Click the **Build** navigation bar icon in the upper right corner of the window. (See Figure 2–2.)

Figure 2–2 Build Navigation Bar Icon

Application Builder appears.

3. Click **Create Application.** (See Figure 2–3.)

Figure 2–3 Create Application

Create Application Wizard appears.

4. Under Select Creation Method, click **Based on Existing Tables** and click **Next**.

   Selecting this option creates a complete application based on existing tables you specify. Selecting the default options results in an application that includes a menu, breadcrumb menus, report page, form page, and a chart page.

5. Select the tables or views on which your application will be based and click **Next**.

   Each application is based on a table or view owned by a specific database schema.

6. Specify an Application Name.

7. If appropriate, select the following options and click **Next**:

   - **Include Breadcrumb Navigation Aids**
   - **Hide Primary Key in Report Page**

8. Confirm your selections and click **Finish**.

   A Confirmation page appears, displaying two icons:

   - Run Application
   - Edit Application

See Also:

- "Running Your Application" on page 2-8 for more information on running your application from the Quick Application Confirmation page
- "Running a Page" on page 7-57 for information on running an application from Application Builder
- "Editing Application Attributes" on page 7-29 for more information on application attributes

Running Your Application

You can run your application by clicking the Run Application icon on the Quick Application Confirmation page.

To run your application from the Quick Application Confirmation page:

1. Click **Run Application**.

   The Login page appears.

2. Log in to your application by typing your workspace username and password and clicking **Login**.

   Your application appears. Note the Developer toolbar at the bottom on the page (See Figure 2–4).

Figure 2–4 Application Builder Developer Toolbar

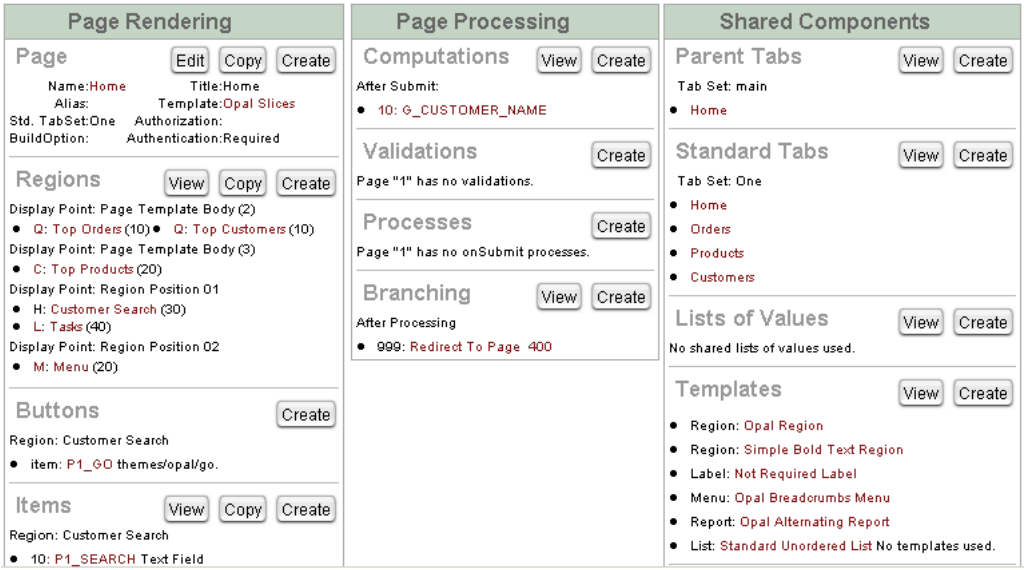
   The Developer toolbar offers a quick way to edit the current page, create a new page, control, or component, view session state, or toggle edit links on an off.

3. Explore your application.

4. To exit your application and return to Application Builder, click **Edit Page** on the Developer toolbar.

   As shown in Figure 2–5, the Page Definition appears.

Figure 2–5 Page Definition



A page is the basic building block of an application. You use the Page Definition to view, create, and edit the components that define a page.

5. To return to Application Builder home page, select the **Application** tab.

See Also:

- "Application Builder Concepts" on page 6-1
- "Understanding the Definition of a Page" on page 7-2
- "Using the Developer Toolbar" on page 7-10

3

Running a Demonstration Application

This section describes how to run and modify the demonstration applications that install with Oracle HTML DB. Running and analyzing how these applications work is an effective way to better understand how you can use Oracle HTML DB to build your own applications.

This section contains the following topics:

- Viewing and Installing a Demonstration Application
- Running a Demonstration Application
- Understanding Sample Application
- Modifying a Demonstration Application
- Viewing Underlying Database Objects

See Also:

- Chapter 1, "What is Oracle HTML DB?"
- Chapter 2, "Quick Start"
- Chapter 6, "Application Builder Concepts"
- Chapter 7, "Using Application Builder"

Viewing and Installing a Demonstration Application

Oracle HTML DB installs with a number of demonstration applications. Use these applications to learn more about the different types of functionality you can include in your applications.

To view the demonstration applications included with Oracle HTML DB:

1. Log in to Oracle HTML DB as described in "Logging in to Oracle HTML DB" on page 2-2.

   Oracle HTML DB appears.

2. From the Tasks list on the right side of the page, select **Review Demonstration Applications**.

   The Demonstration Applications page appears, displaying links to the following applications:

   - *Sample Application* offers a working demonstration that highlights basic design concepts
   - *Collection Showcase* demonstrates shopping cart concepts
   - *Web Services* serves an example of how you can use Web Services
   - *Presidential Inaugural Addresses* demonstrates Oracle Text

     See Also: "Implementing Web Services" on page 12-15

The Status column on the Demonstration Applications page indicates whether or not an application is currently installed.

To re-install a demonstration application:

1. Navigate to the Demonstration Applications page as described in the previous procedure.
2. Scroll down to the application you wish to install, click **Re-install**.
3. Follow the on-screen instructions.

Running a Demonstration Application

Oracle HTML DB installs with a number of demonstration applications. Once a demonstration application has been installed, there are a number of ways to run it.

Running an Application from Demonstration Applications

The simplest way to run a demonstration application is navigate to the Demonstration Applications page.

To run a demonstration application from the Demonstration Applications page:

1. Log in to Oracle HTML DB as described in "Logging in to Oracle HTML DB" on page 2-2.

   Oracle HTML DB appears.

2. From the Tasks list on the right side of the page, select **Review Demonstration Applications**.

   The Demonstration Applications page appears.

3. Locate the application you wish to run.

4. In the Action column, click **Run**.

5. Enter your Oracle HTML DB username and password and click **Login**.

Running an Application from Application Builder

You can also run a demonstration application from Application Builder. Application Builder is the tool you use to build the pages that comprise an application.

To run a demonstration application from Application Builder:

1. Log in to Oracle HTML DB as described in "Logging in to Oracle HTML DB" on page 2-2.

   Oracle HTML DB appears.

2. Click the **Build** navigation bar icon in the upper right corner of the window.

   Application Builder appears.

3. From the Available Applications list, select the desired demonstration application and click **Go**.

4. Click **Run**. (See Figure 3–1.)

Figure 3–1 Run Icon

Run

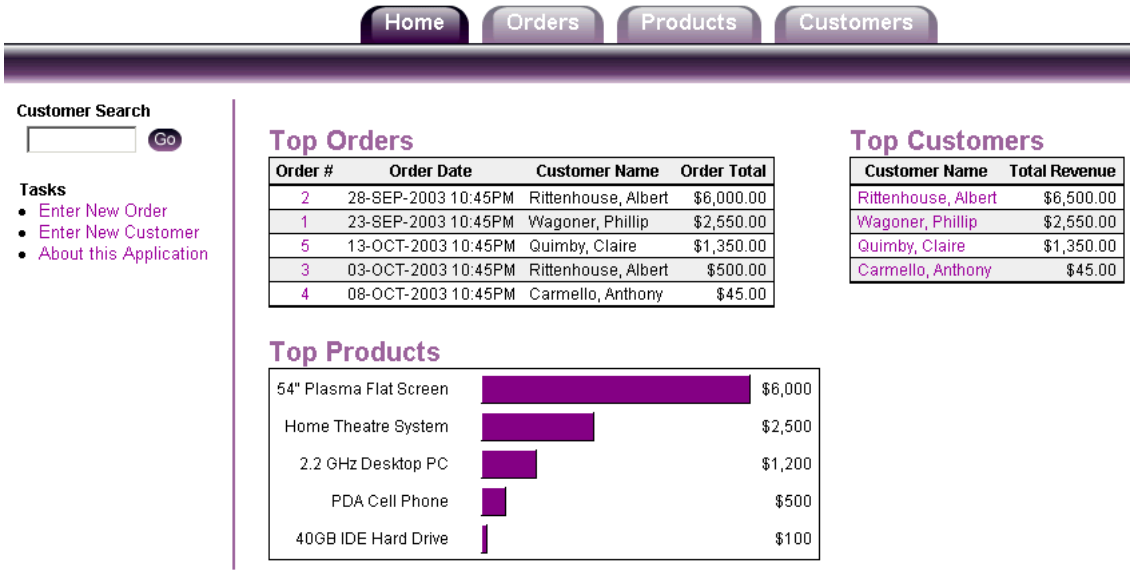
5. Enter your Oracle HTML DB username and password and click **Login**.

Understanding Sample Application

Each demonstration application features a different set of functionality. This section describes the demonstration application, *Sample Application*.

As show in Figure 3–2, Sample Application features an easy-to-use interface for viewing, updating, and searching order and customer information for electronic and computer products. Users can navigate between the pages using the Home, Orders, Products, and Customers tabs.

Figure 3–2 Sample Application



Sample Application demonstrates the following functionality:

- Searching for customers
- Viewing order and customer details
- Editing customer and product information
- Viewing all orders, products, or customers
- Sorting order, product, and customer information by column heading
- Creating new orders, products, and customers

- Viewing pages in printer friendly mode

The sections that follow describe specific functionality available on each page.

See Also: "What is a Page?" on page 6-2

About the Home Page

The Home page contains three distinct sections:

- Customer search field
- Tasks list
- Orders, customers, and products reports

You can search for customers using the Customer Search field. Type a customer name in the Customer Search field and click **Go**.

The right side of the Home page features two different report formats and a bar chart. Note that in the Top Order report you can link to order details by selecting the Order number. In the Top Customers report you can link to customer details by selecting the customer name.

The Tasks list consists of a series of links that take you to other pages within the application. Links available on the Home Page Tasks list include:

- **Enter New Order** links to a wizard that walks you through the process of creating a new order. First you select a customer name and then you add items to the order
- **Enter New Order** links to a form where you can enter new customer information.
- **About this Application** links to an informational page describing this application.

About the Orders Page

Use the Orders page to search, view, and enter order information. By default, the right side of the page displays current orders. To search for an order, enter the order number in the Search for field and click **Go**. Click a column heading to sort the information. You view more details about a specific order by clicking the view icon. (See Figure 3–3.)

Figure 3–3 View Icon

To enter new orders or view reports that display order revenue by month and orders by customer, use the Tasks list on the left side of the page.

About the Products Page

Use the Products page to view and edit product information. By default, the right side of the page displays current products. Click a column heading to sort the information. You can edit a product description by clicking **Edit**. To add a new product or view a chart displaying products by category, use the Tasks list on the left side of the page.

About the Customers Page

Use the Customers page to view and edit customer information. To search for a customer, type a customer name in the Search for field and click **Go**. By default, the right side of the page displays customer information. Click a column heading to sort the information. Click **Edit** to update customer information. To view all customers or to enter a new customer record, use the Tasks list on the left side of the page.

Viewing Pages in Printer Friendly Mode

Clicking Print in the upper right corner of the page displays the current page in Printer Friendly mode. When in Printer Friendly mode, the HTML DB engine displays all text within HTML from fields as text.

To enable your application to display in Printer Friendly mode, you need to create and then specify a Print Mode Page Template on the Edit Application Attributes page.

> **See Also:** "Working with Templates" on page 7-16 and "About User Interface Templates" on page 7-34 for more information on specifying a Print Mode Page Template

Modifying a Demonstration Application

Once you understand the type of functionality available in a demonstration application, the next step is to learn more about how each page is constructed. You

edit an application using Application Builder. Using Application Builder you can edit existing pages in an application, add pages to an application, or create entirely new applications.

About the Developer Toolbar

When you log in to Oracle HTML DB having developer privileges and run an application, a Developer toolbar displays at the bottom of every page. As shown in Figure 3–4, the Developer toolbar offers a quick way to edit the currently running page, create a new page, control, or component, view session state, or turn edit links on or off.

Figure 3–4 Developer Toolbar in Sample Application

The Developer toolbar consists of the following links:

- **Edit Application** links you to the Application Builder home page. (See "Viewing a Page Definition" on page 7-5.)
- **Edit Page** accesses the Page Definition for the currently running page. (See "Viewing Page Attributes" on page 7-37.)
- **New** links to a wizard that enables you to create a new blank page, a component (report, chart, or form), a page control (region, button, or item), or a shared component (menu, list, or tab).
- **Session** links you to session state information for the current page. (See "Viewing Session State" on page 6-13.)
- **Debug** runs the current page in debug mode. (See "Accessing Debug Mode" on page 9-2.)
- **Show edit links** toggles between Show edit links and Hide edit links. Clicking **Show edit links** displays an edit link (resembling four gray dots) to the right of most page components or controls. By clicking an edit link you can edit the selected component or control.

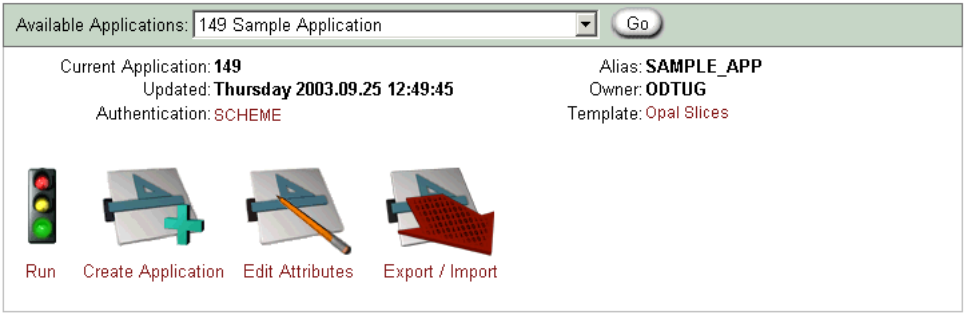
Editing a Demonstration Application

There are two common ways to edit a demonstration application:

- From Demonstration Applications page, click **Edit** next to the desired application
- If you are running an application, click **Edit Application** on the Developer toolbar

As shown in Figure 3–5, the Available Applications list displays at the top of the Application Builder home page. Note that the Available Applications list displays the current application name.

Figure 3–5 Available Applications List

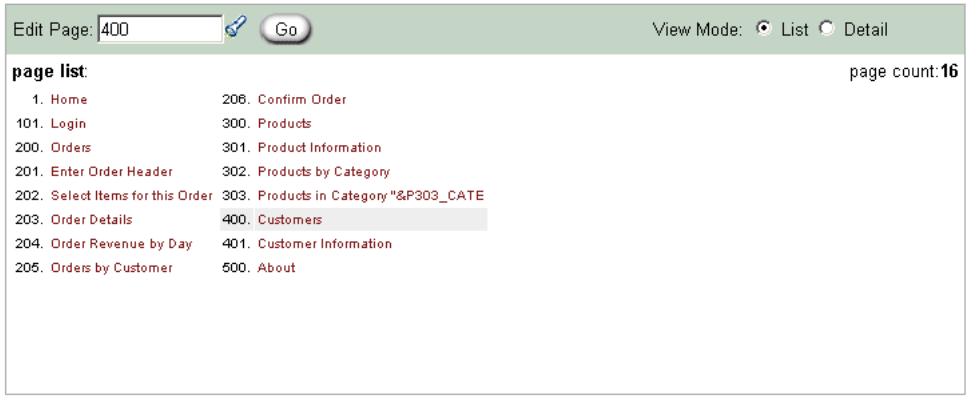


The current application ID, last update date, authentication scheme, alias, owner, and selected template display directly beneath the list. You can run an existing application, create a new application, edit application attributes, or export information by clicking the following icons:

- **Run** submits the pages in the current application to the HTML DB engine to render viewable HTML beginning on the Home Link identified in the application attributes.
- **Create Application** creates a new application using the Create Application Wizard.
- **Edit Attributes** displays the Edit Application Attributes page.
- **Export/Install** links you to the Export Import Wizard.

As shown in Figure 3–6 on page 3-9, the bottom of the Application Builder home page displays a list of all pages in the currently selected application.

Figure 3–6 Page List View



To access a specific page, enter a page ID in the Edit Page field and click **Go**. To edit a page, drill down on the page name. Note that application in Figure 3–6 contains 16 pages.

See Also:

- "Accessing Application Builder" on page 7-2 for more information on using the Application Builder home page
- "Viewing a Page Definition" on page 7-5 for more information on viewing, creating, and editing the components and controls that define a page

Viewing Underlying Database Objects

The HTML DB engine renders applications in real time based on data stored in database tables. You can view the database objects for any demonstration application in SQL Workshop.

See Also: "Using SQL Workshop to Manage Database Objects" on page 5-1

To view the database objects used for an application:

1. Click the **SQL** icon.

   You view by schema and type and then by name. Under Data Browser, you can selecting existing database objects by selecting a database object type.

2. Under Data Browser, select **Tables**.
3. To create a search:
   - In Schema, select your workspace
   - In Type, select **Table**
   - In Search, type **DEMO**
   - Click **Go**

   All tables having names that contain the string `DEMO` appear.
4. To view table details, click the view icon adjacent to the appropriate table name.

   The Object Detail page appears.
5. Optionally, select a task from Tasks list on the right side of the page.

Part II

Using Oracle HTML DB

Part II describes how to use Data Workshop, SQL Workshop, and Application Builder to develop database-driven applications.

Part II contains the following chapters:

4

Managing Data with Data Workshop

This section describes how to use Data Workshop to import data into and export data from your hosted database.

This section contains the following topics:

- About Data Workshop
- Importing Data
- Exporting Data

See Also:

- Chapter 1, "What is Oracle HTML DB?"
- Chapter 2, "Quick Start"

About Data Workshop

Oracle HTML DB renders information stored in a Oracle database to create a collection of database-driven Web pages called an **application**. Using Data Workshop you can import data into and export data from the hosted database. Supported import formats include:

- Text such as comma or tab delimited data
- XML documents
- Spreadsheets

Supported export formats include:

- Text such as comma or tab delimited data
- XML documents

To access Data Workshop:

1. Click the **Data** icon. (See Figure 4–1.)

Figure 4–1 Data Icon

2. Under Data Import and Data Export, click the appropriate link.

Importing Data

You can use Data Workshop to import text files, XML documents, and data stored in a spreadsheet into an Oracle database.

Topics in this section include:

- Importing a Text File
- Importing an XML Document
- Importing Spreadsheet Data

Importing a Text File

For files less than 30KB, you can copy and paste tab delimited data directly into the Import Text Wizard. For files larger than 30KB, you must upload a separate file.

To load a text file:

1. Click the **Data** icon.
2. Under Data Import, click **Import Text Data**.

   The Import Text Data Wizard appears.
3. Under Import to, select **Existing table** or **New table**.
4. Under Import from, select **Upload file** or **Copy and paste**.
5. Follow the on-screen instructions.

Importing an XML Document

Data Workshop supports the import of XML documents adhering to the Canonical XML specification.

To import an XML document:

1. Click the **Data** icon.
2. Under Data Import, click **XML Data**.

   The XML Import Wizard appears.
3. Follow the on-screen instructions.

Importing Spreadsheet Data

You can load spreadsheet data by either copying and pasting text, or by importing a file. To copy and paste text, the spreadsheet file must be less than 30KB. For files larger than 30KB, you can import the file in a delimited format (such as comma delimited (.csv) or tab delimited), upload the file, and then load the data into a new or existing table.

To import spreadsheet data:

1. Click the **Data** icon.
2. Under Data Import, click **Import Spreadsheet Data**.

   The Spreadsheet Data Import Wizard appears.
3. Under Import to, select **Existing table** or **New table**.
4. Under Import from, select **Copy and paste** or **Upload file**.
5. Follow the on-screen instructions.

Exporting Data

You can also use Data Workshop to export the contents of a table to a text file or XML document.

Topics in this section include:

- Exporting to a Text File
- Exporting to an XML Document

Exporting to a Text File

Use the Text Export Wizard to export the contents of a table to a text file. For example, you could export an entire table to a comma delimited file (.csv).

To export a table to a text file:

1. Click the **Data** icon.
2. Under Data Export, click **Export Text Data**.

   The Text Data Export Wizard appears.
3. Follow the on-screen instructions.

You select the schema and choose the table and columns to be exported. You can also specify the type of separator to be used to separate column values as well as whether column text strings are identified using single or double quotation marks.

Exporting to an XML Document

Use the XML Export Wizard to export the contents of a table to an XML document adhering to the Canonical XML specification.

To export a table to an XML document:

1. Click the **Data** icon.
2. Under Data Import, click **XML Export**.

   The XML Export Wizard appears.
3. Follow the on-screen instructions.

You select the schema and choose the table and columns to be exported.

5

Using SQL Workshop to Manage Database Objects

This section provides information on how to use SQL Workshop to view and manage database objects as well as browse the data dictionary.

This section contains the following sections:

- About SQL Workshop
- Viewing Database Objects
- Managing Database Objects
- Managing User Interface Defaults
- Browsing the Data Dictionary

See Also:

- Chapter 1, "What is Oracle HTML DB?"
- Chapter 2, "Quick Start"

About SQL Workshop

You can use SQL Workshop to view and manage database objects from a Web browser. SQL Workshop includes the following navigation tabs:

- **SQL Workshop.** Offers quick access to the SQL Command Processor, SQL Script Repository, and Database Browser. Includes links to the Query By Example, Generate DDL (data definition language), and Create Table wizards.
- **SQL.** Access the SQL Command Processor to run SQL or PL/SQL statements.
- **Scripts.** Access the SQL Script Repository and Control Files Repository.

- **Browse.** Use the Database Browser to view existing database objects and the data dictionary.

To access SQL Workshop:

1. Click the **SQL** icon. (See Figure 5–1).

Figure 5–1 SQL Icon

2. Under SQL Workshop, select the appropriate link, or click the **SQL**, **Scripts**, or **Browse** tabs at the top of the page.

About Transaction Support

Oracle HTML DB is a browser based development environment which communicates over HTTP. Because HTTP is a stateless protocol, any command you issue using SQL Workshop is automatically followed by a database COMMIT. There is no support for transactions that span multiple pages in the SQL Workshop. For example, it not possible issue an `UPDATE` statement on one page in the SQL Workshop and then revert it on a subsequent page using a `ROLLBACK` command.

Since the commands `COMMIT`, `ROLLBACK` and `SAVEPOINT` are executed as one transaction, you can include these commands in SQL Workshop by using scripts

> **See Also:** Using the SQL Script Repository on page 5-8 for information on running scripts

About Support for SQL*Plus Commands

SQL Workshop does not support SQL*Plus commands. If you attempt to enter a SQL*Plus command in SQL Workshop an error message displays. The following are examples of unsupported SQL*Plus commands:

```
SET ECHO OFF
SET ECHO ON
SET VERIFY ON
SET LONG 600
```

```
COLUMN dummy NOPRINT
COLUMN name FORMAT A20
DEFINE
ACCEPT
PROMPT
REMARK
SHOW
```

Viewing Database Objects

You can use SQL Workshop to view database objects. For example, you can view details about database objects by querying the Oracle dictionary. You can also run SQL commands and SQL scripts in the SQL Command Processor or view database objects in the Database Browser.

Topics in this section include:

- Using the SQL Command Processor
- Browsing Database Objects
- Viewing Database Objects by Object Type

Using the SQL Command Processor

You can use the SQL Command Processor to run SQL commands and SQL scripts on any Oracle database schema for which you have privileges.

To use the SQL Command Processor:

1. Click the **SQL** icon and select the **SQL** tab.

   The SQL Command Processor appears.

2. Select a schema from the list and follow the on-screen instructions.
3. To run entered commands, click **Run SQL**.
4. To save entered commands, click **Save**.

> **See Also:** "Accessing Saved Commands in the SQL Archive" on page 5-11 for more information on viewing saved commands and queries

About Command Termination

You can terminate commands in the Command Processor using either a semicolon (;) or forward slash (/). Consider the following examples:

```
INSERT INTO emp
      (50,'John Doe','Developer',10,SYSDATE,1000,10);

INSERT INTO emp
      (50,'John Doe','Developer',10,SYSDATE,1000,10)
/
```

The first example demonstrates the use of a semicolon (;). The second example demonstrates the use of forward slash (/).

Using Explain Plan

Use the Explain Plan link to view the plan the Oracle Optimizer uses to run your SQL Command.

To view the Explain Plan:

1. Click the **SQL** icon.
2. Select the **SQL** tab and then **Plan**.

   Explain Plan appears.
3. Enter a command in the field provided and click **Explain Plan**.

Browsing Database Objects

You can use the Data Browser to view database objects. To find a database object, select the schema you would like to view. The values available in the schema depend upon your resource privileges.

To browse database objects:

1. Click the **SQL** icon and select the **Browse** tab.

   The Data Browser appears.
2. To view details about a specific object, click the view icon.

To search for database objects:

1. Click the **SQL** icon.
2. Select the **Browse** tab and then **Search Objects**.

3. Enter search criteria in the fields provided and click **Go**.

You can search for columns within tables or text within source code by entering a search string in the Search field and selecting a Search Option at the top of the page. Searches are case insensitive and no wildcards or quotes are necessary.

Querying by Example

Once you have located a specific table you can query the Oracle data dictionary to discover more details.

To Query by Example:

1. Click the **SQL** icon and select the **Browse** tab.

   The Data Browser appears.

2. To view details about a specific object, click the view icon.

   The Object Detail appears.

3. Select **Query by Example** from the Tasks list.
4. Follow the on-screen instructions.

Viewing Database Objects by Object Type

You can also use the Data Browser to view database objects by type.

To view database objects by object type:

1. Click the **SQL** icon.
2. Under the **Data Browser**, select an object type.

   The Data Browser appears.

3. To view details about a specific object, click the view icon.
4. Follow the on-screen instructions.

Managing Database Objects

You can use SQL Workshop to manage database objects. For example, you can create new database objects, manage script files and control files, or alter a table.

Topics in this section include:

- Browsing Database Objects

Browsing Database Objects

You can use the Data Browser to view existing database objects.

To view or edit existing database objects:

1. Click the **SQL** icon.
2. s, select the type of database object you would like to view.
3. To search for an object, select a schema, an object type, type a search string in the Search field, and click **Go**.
4. To view object details, click the view icon adjacent to the appropriate name.
5. Optionally, select a task from the Tasks list on the right side of the page.

Creating Database Objects

You can create new database objects using the Create Database Object Wizard.

To create new database objects in SQL Workshop:

1. Click the **SQL** icon.
2. From the Tasks list on the right side of the page, select **Create a database object**.

   The Create Database Object Wizard appears.
3. Follow the on-screen instructions.

Dropping Database Objects

You can drop database objects using the Drop Database Object Wizard. When you drop a table using this wizard, you also remove all related triggers and indexes.

To drop a database object:

1. Click the **SQL** icon.
2. From the Tasks list on the right side of the page, select **Drop database object**.

   The Drop Database Object Wizard appears.
3. Select a schema and then an object type.
4. Follow the on-screen instructions.

Restoring Dropped Database Objects

If you are running Oracle HTML DB with an Oracle 10*g* database, you can use the Recycle Bin to view and restore dropped database objects. When you drop a table, the space associated with the table is not immediately removed. Oracle renames the table and places it and any associated objects in the Recycle Bin where it can be recovered at a later time.

> **Note:** The Recycle Bin feature is only available if you are running Oracle HTML DB with an Oracle 10g database.

To use the Recycle Bin:

1. Click the **SQL** icon.
2. Under SQL Workshop, click **Recycle Bin**.

   The Recycle Bin appears.
3. To search for an object, select a schema, an object type, type a search string in the Search field, and click **Go**.
4. To view object details, click the view icon adjacent to the appropriate name.
5. On the Object Summary page you can:
   - Click **Restore Object** to restore the current object
   - Click **Purge** to permanently delete the current object

To empty the Recycle Bin without viewing the objects:

1. Click the **SQL** icon.
2. Under SQL Workshop, click **Recycle Bin**.

   The Recycle Bin appears.
3. From the Tasks list on the right side of the page, select **Purge Recycle Bin**.

Using the SQL Script Repository

You can use the SQL Script Repository to view, edit, and run uploaded script files. For example, you can upload new script file as well as create and edit your create table, create index, and create PL/SQL package scripts.

Topics in this section include:

- Managing Script Files in the SQL Script Repository
- Uploading and Creating Script Files
- Using Parameters in a Script
- Including SQL Queries in a Script
- Exporting a Script File

Managing Script Files in the SQL Script Repository

To view scripts in the SQL Script Repository:

1. Click the **SQL** icon and select the **Scripts** tab.

   SQL Script Repository appears. Scripts are stored based on HTML DB username.
2. To search for a script, select a username from the Show list, enter a search string in the Find field (optional), and click **Go**.
3. While in the Script Repository you can:
   - Reorder a list by clicking the column heading
   - View details about a specific file by clicking the view icon
   - Edit a script by clicking the edit icon
   - Parse a script to be run by clicking **Parse**
   - Run a script by clicking **Run** in the Actions column
   - Delete a script by selecting it and clicking **Delete Checked**

- Upload a script by clicking **Upload**
- Create a script by clicking **Create**

To view script details:

1. In the Script Repository, click the view icon.

   The Script - Files Details page appears.

2. Under View Links, you can:
   - Click **Native file format** to download the file locally
   - Click **View document as text** to view the file in your Web Browser
   - Click **Parse this script** to parse the script to run

To run a script in the Script Repository:

1. In the Actions column, click **Run**.

   The Script - Run page appears.

   If you have parameters in your script you must define them. You can define up to ten different parameters in each script.

2. Enter a parameter name and value in the fields provided.
3. To view the script file, click **View File**.
4. To run the script file, click **Run Script**.

   The Script - Run Results page appears displaying the number of success, failure and the elapsed time. RED indicates that errors occurred while executing the file.

5. To view the script file source, click **View Source**.
6. To run the file again, click **Run Script** in the left navigation pane.

   Once you have run a script file, you can view a history of previous executions by clicking **Previous Runs** on the Script - Run page.

To delete a script file from the Script Repository:

1. In the Script Repository, select the script to be deleted.
2. Click **Delete Checked**.

Uploading and Creating Script Files

To upload a script file into the Script Repository:

1. In the Script Repository, click **Upload**.

   The Upload Script page appears.

2. Follow the on-screen instructions.

If the script file you upload has a valid file extension, SQL Workshop recognizes the file as a script and automatically parses it. Table 5–1 describes the file extensions SQL Workshop considers to be valid for script files.

Table 5–1 Valid Script File Extensions

Extension	Description
pkh	Package headers
plb	Package bodies
sql	Scripts
con	Constraints
ind	Indexes
sqs	Sequences
tab	Tables
trg	Triggers
pkb	Package bodes
pks	Package specs

To create a script file while in the Script Repository:

1. In the Script Repository, click **Create**.

   The Create Script page appears.

2. Follow the on-screen instructions.

Using Parameters in a Script

You can parameterize a script using a pound (#) or ampersand (&). The following two examples demonstrate valid parameter syntax.

```
CREATE TABLE #OWNER#.xyz (X INT)
/
```

```
CREATE TABLE #OWNER#.abc (Y NUMBER)
/

CREATE TABLE &OWNER.xyz (X INT)
/
CREATE TABLE &OWNER.abc (Y NUMBER)
/
```

Including SQL Queries in a Script

If you include SELECT statements in a script, the script will run without errors, but the result set will not display.

Exporting a Script File

You can export SQL Script Repository scripts using the Export Import Wizard in Application Builder.

To export a script from SQL Workshop:

1. Click the **Build** icon.
2. When Application Builder appears, click **Export/Import**.

   The Export/Import Wizard appears.
3. Select **Export** and click **Next**.
4. Click the **Export Script** tab and follow the -screen instructions.

If you export a UNIX format, the wizard generates a file with rows delimited by `CHR (10)` (that is, line feeds). If you export a DOS format then each row is terminated with `CHR(13)||CHR(10)` (that is, `CR LF` or carriage return line feed).

> **See Also:** "Exporting an Application and Related Files" on page 10-4

Accessing Saved Commands in the SQL Archive

When you click Save in SQL Command Processor, SQL Workshop saves entered commands and scripts to the SQL Archive.

SQL Archive is different from SQL Script Repository. By saving frequently used SQL commands in the SQL Archive, you can run the commands again without retyping. When you save a SQL command to SQL Archive, the saved command does not appear in Script Repository.

To view the SQL Archive:

1. Click the **SQL** icon.
2. Select the **SQL** tab and select **Archive**.

   The SQL Archive appears.
3. Follow the on-screen instructions.

Accessing the SQL Command History

SQL Command History displays the 200 most recent commands and scripts run in the SQL Command Processor.

To view the SQL Command History:

1. Click the **SQL** icon.
2. Select the **SQL** tab and select **History**.

   SQL Command History appears.
3. To run a command again, click the appropriate link.

   The SQL command or the script displays in the SQL Command Processor.

Generating DDL

You can use DDL statements to create, alter, and drop schema objects when they are no longer needed. You can also use DDL statements to grant and revoke privileges and roles, to analyze table, index, or cluster information, to establish auditing options, or to add comments to the data dictionary.

To generate DDL statement in SQL Workshop:

1. Click the **SQL** icon.
2. Under SQL Workshop, click **Generate DDL**.

   The Generate DDL Wizard appears.
3. Follow the on-screen instructions.

See Also:

- *Oracle Database SQL Reference* for more information on DDL statements
- *Oracle Database Concepts* for more information on the data dictionary

Managing Control Files

Control files enable you to run a series of scripts in a predefined order. From the Control Files Repository, you can create, edit, delete or run control files.

To access the Control Files Repository:

1. Click the **SQL** icon.
2. Select the **Scripts** tab and then **Control Files**.

   Control Files Repository appears.
3. To search for a script, select a username from the Show list, enter a search string in the Find field (optional), and click **Go**.
4. While in the Control Files Repository you can:
   - Reorder a list by clicking the column heading
   - Edit a file by clicking the edit icon
   - Run a file by clicking **Run**
   - Delete a script by selecting it and clicking **Delete Checked**

To create a control file:

1. In the Control Files Repository, click **Create**.

   The Control File Create page appears.
2. Enter a name for the control file, select the script files you would like to include, and click **Create**.
3. Specify the execution order and click **Done**.

To edit a control file:

1. In the Control Files Repository, click the edit icon.

   The Edit File page appears.
2. On the Edit File page you can:

- Change the order in which files are executed by clicking **Edit Execution Order**
- Add additional script files by clicking **Add More Files**
- Delete a script by selecting it and clicking **Delete Checked**

To run a control file in the Control Files Repository:

1. In the Action Column, click **Run**.

   The Run File page appears.

2. Select an Oracle Schema from the Parse As list.

   If your script files include parameters you must define them. You can define up to ten different parameters in each script.

3. Enter a parameter name and value in the fields provided.

4. Click **Run File**.

   The Run Results page appears displaying the number of success, failures, and the elapsed time. RED indicates that errors occurred while executing the file.

   If you wish to run the control file in the background, select **Run in Background**. Running a control file in the background means that Oracle HTML DB submits it as a job. The advantage of this approach is that you do not have to wait for it to finish to continue using SQL Workshop.

5. To run the file again, click **Run File** in the left navigation pane.

Viewing the Control File Run History

Once you have run a file, you can view a history of previous executions by clicking **Previous Runs** on the Run File page.

To view the Control File Run History:

1. Run the control file as described in the previous procedure.
2. Click **Run Script** in the left navigation pane.
3. On the Script - Run page, click **Previous Runs**.

Viewing Control File Job Status

You can view control file job status from either the Edit File or Run File page.

To view the status of a job:

1. Run the control file as described in the previous section with the **Run in Background** option selected.
2. Click **Job Status** in the left navigation pane.

Managing Tables

You can also use SQL Workshop to create new tables or edit existing tables.

To create a new table:

1. Click the **SQL** icon.
2. Under SQL Workshop, click **Create Object**.

   The Create Table Wizard appears.
3. Select an object and click **Next**.
4. Follow the on-screen instructions.

To edit an existing table, you must first navigate to it using the Data Browser.

To edit an existing table:

1. Click the **SQL** icon.
2. Under Data Browser, select **Tables**.
3. To search for a table, select a schema, table type, type a search string in the Search field, and click **Go**.
4. To view table details, click the view icon adjacent to the appropriate table name.
5. Select task from the Tasks list on the right side of the page.

Managing User Interface Defaults

UI (user interface) defaults enable developers to assign default user interface properties to a table, column, or view within a specified schema. When a developer creates a form or report using a wizard, the wizard uses this information to create default values for region and item properties.

Note that UI defaults are associated with a table and can be used in an application created from a form and report wizards. This means that you cannot use shared list of values to define UI defaults, since shared list of values are associated with a single application.

Topics in this section include:

- Managing Tables Using UI Defaults
- Applying UI Defaults to a Table or View
- Exporting UI Defaults

> **See Also:** "Application Builder Concepts" and "Using Application Builder" for more information on regions and item properties

Managing Tables Using UI Defaults

To view tables using UI Defaults:

1. Click the **SQL** icon.
2. Select the **Browse** tab and then **UI Defaults**.
3. To edit the UI Defaults associated with a specific table, click the edit icon next to the table name.

   The following table-level UI defaults display at the top of the page:

   - **Schema** is the schema that owns the table.
   - **Table Name** is the name of the selected table.
   - **Title** is a modified version of **Table Name** in which the first letter is capitalized and any underscores are replaced with spaces.

   Column-level UI Defaults appear next. You can edit attributes for all displayed columns, by clicking **Grid Edit**.
4. To edit a specific column, click the edit icon adjacent to the column name.

   The Edit Column-level UI Defaults page appears.

The top of the Column-level UI Defaults page displays the table and column name. **Column Name** is the name of the selected column. Use **Default for Label** to specify a label for reports and forms. By default, this field displays a modified version of Column Name in which the first letter is capitalized and any underscores are replaced with spaces. Default attributes for reports and forms appear next.

Available **Display for Reports** attributes include:

- **Display** - Indicates whether the column displays in a report. The default is **Yes**.
- **Display Seq** - Specifies the display sequence of items in a report. The default value is based on the column ID, which is based on the order of the columns in the table.

- **Mask** - Indicates if a mask should be applied against the data. Not used for character based items.
- **Alignment** - Specifies report alignment (Left, Center, or Right). If the column is a number, the default is **Right**. Otherwise, the default is **Left**.
- **Searchable** - Indicates whether the column should be searchable in reports. If the column is `VARCHAR2` or `CHAR`, the default is **Yes**. If not, the default is **No**.
- **Group By** - Indicates whether this column should be used for Group By and then the sequence of the grouping. The default is **Yes**.

Available **Defaults for Reports** attributes include:

- **Display** - Indicates whether this column displays in a form. The default is **Yes**.
- **Display Seq** - Specifies the sequence of items in a form. The default is based on the column ID, which is based on the order of the columns in the table.
- **Display As** - Indicates how items in a form display. The default selection is **Text Field**.
- **Mask** - Indicates a mask to be applied against the data in a form. Not used for character based items.
- **LOV Query** - Generates a LOV (list of values). Only valid for certain Display As types.
- **Default Value** - Specifies the default value associated with this column.
- **Width** - Specifies the display width.
- **maxWidth** - Specifies the maximum string length a user is allowed to enter in this item.
- **Height** - Specifies the display height of an item.
- **Required** - Used to generate a validation in which the resulting item must not be null. If resulting item is not null, select **Yes**.
- **Help Text** - Becomes Item help. By default, this text is pulled from the column hint (if applicable).

Applying UI Defaults to a Table or View

You can view a listing of tables without UI Defaults on the Default Table & Column-level UI Defaults page.

To view the Default Table & Column-level UI Defaults page:

1. Click the **SQL** icon.
2. Select the **Browse** tab and then **UI Defaults**.
3. From the Tasks list, select **Apply UI Defaults**.
4. Follow the on-screen instruction.

Exporting UI Defaults

When you export UI Defaults, all UI Defaults for the selected schema are exported to a single SQL*Plus script. When prompted by your browser, save this file to your hard drive. The file contains an API call to create table hints by making calls to the application PL/SQL API. All work is performed in a single transaction.

To export UI Default HInts:

1. Click the **SQL** icon.
2. Select the **Browse** tab and then **UI Defaults**.
3. From the Tasks list, select **Export UI Defaults**.
4. Follow the on-screen instruction.

Browsing the Data Dictionary

Each Oracle database has a data dictionary. An Oracle data dictionary is a set of tables and views that are used as a read-only reference about the database. For example, a data dictionary stores information about both the logical and physical structure of the database. A data dictionary also stores information about valid Oracle database users, integrity constraints for tables in the database, and the amount of space allocated for a schema object as well as how much of it is being used.

To browse the data dictionary:

1. Click the **SQL** icon
2. Select the **Browse** tab and then **Data Dictionary Browser**.

   The Data Dictionary Browser appears.
3. Click the view icon to display the Query By Example (QBE) form. Use this form to query the Oracle data dictionary for details about database objects.

See Also: *Oracle Database Concepts* for more information on the data dictionary

To view data dictionary reports:

1. Click the **SQL** icon
2. Select the **Browse** tab and then **Dictionary Reports**.

   The Data Dictionary Reports page appears.
3. Make a selection from the list and follow the on-screen instructions.

6

Application Builder Concepts

This section provides basic conceptual information about Application Builder. Application Builder is the core component within Oracle HTML DB that enables you to build database centric Web applications.

This section contains the following topics:

- About Page Rendering and Page Processing
- How Page Processing and Page Rendering Work
- Understanding Session State Management
- Managing Session State Values
- Understanding URL Syntax
- Using Substitution Strings

See Also:

- Chapter 1, "What is Oracle HTML DB?"
- Chapter 7, "Using Application Builder"

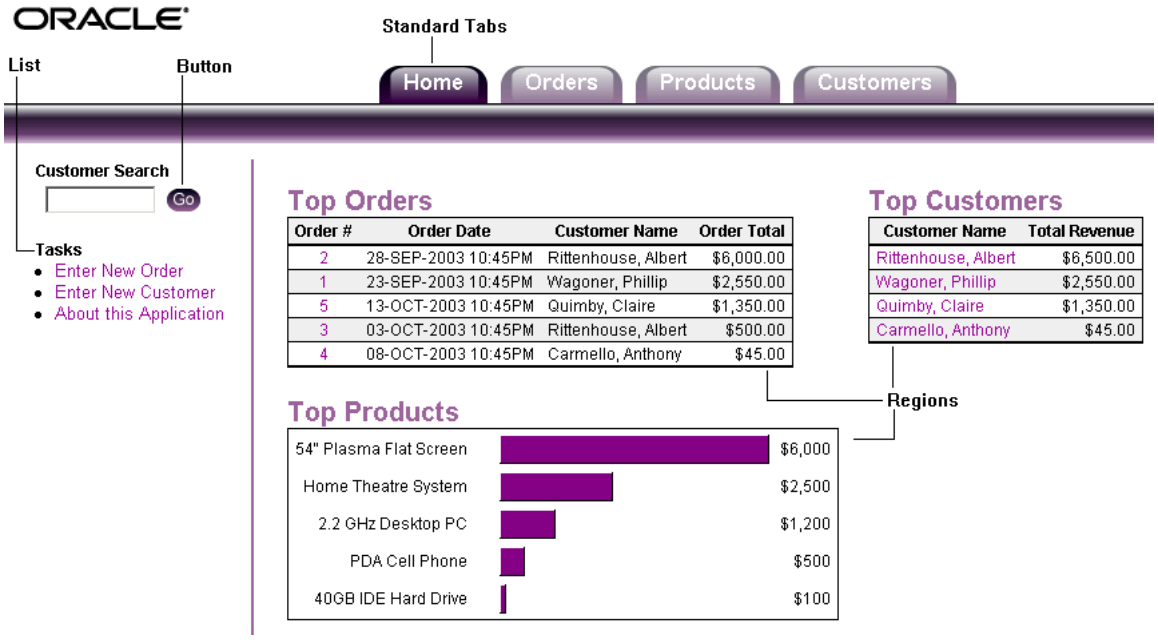
About Page Rendering and Page Processing

In Oracle HTML DB you use Application Builder to build dynamically rendered applications. An application is a collection of database-driven Web pages. You can link pages together using tabs, buttons, or hypertext links. Each page can have buttons and items and can include application logic. You can branch from one page to the next using conditional navigation, perform calculations, validations, and display reports, calendars, and charts. You can generate reports, charts, and forms using built-in wizards, static HTML, or deliver more custom rendering with PL/SQL programming.

What is a Page?

A page is the basic building block of an application. When you build an application in Application Builder you can include a number of common user interface elements, including standard tabs, navigation bar icons, buttons, items, and regions. Figure 6–1 illustrates the use of some of these elements.

Figure 6–1 Sample Application



You can view all the elements that make up a page by accessing the Page Definition.

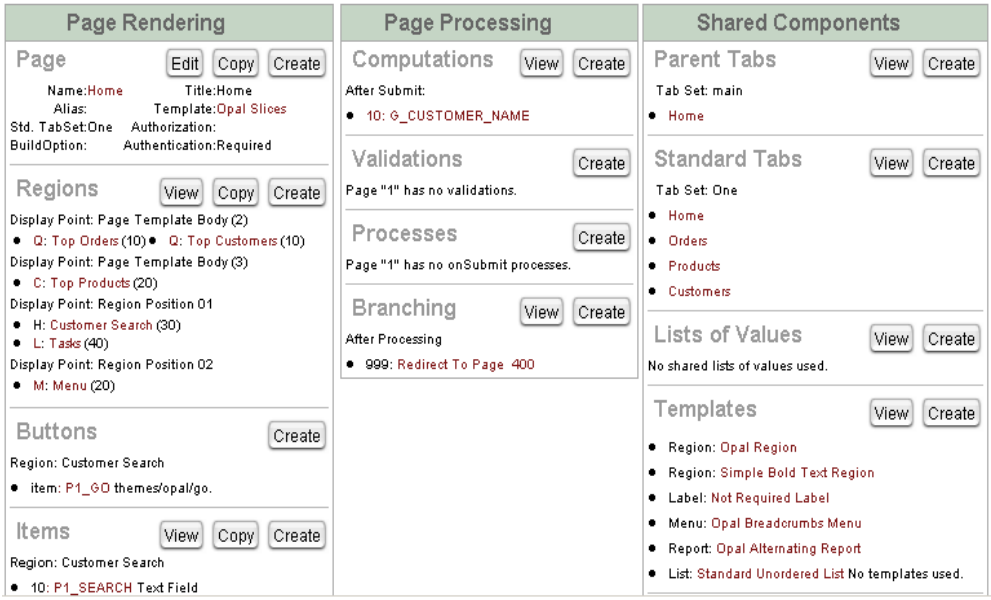
To view the Page Definition for an existing page:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages appears at the bottom of the page.
3. To edit a specific page, enter the page ID in the Edit Page field and click **Go**, or click the page name.

   The Page Definition appears. (See Figure 6–2.)

Figure 6–2 Page Definition



By default, the Page Definition is divided into three sections:

- Page Rendering
- Page Processing
- Shared Components

The left section, **Page Rendering**, lists page level attributes as well as any UI controls and logic that is executed at the time the page is rendered. The middle section, **Page Processing**, lists the logic controls (such as computations and processes) that are evaluated and executed when the page is processed. The far right section, **Shared Components**, lists common components that display on every page within an application.

The following list briefly describes Page Rendering components:

- **Page**. Defines page level attributes such as the page name, title and template.
- **Regions**. Defines regions. A region is an area of a page that uses a specific template to generate HTML content. Each page can have any number of regions. You can use regions to group other controls, such as buttons and

items, together. You can create simple regions that do not generate additional HTML, or create elaborate regions that frame content within HTML tables or images. The HTML DB engine displays regions in sequence within columns. You can choose whether a region displays conditionally.

- **Buttons**. Lists the buttons on the current page. Buttons are used to submit a page. When you submit a page, the HTML DB engine processes it, or redirects users to another page without any processing. Buttons can be implemented as an HTML button, an image, or by using a template.
- **Items**. Lists the items grouped by region. Items are HTML form elements such as text fields, select lists and check boxes with an associated session state.
- **Computations**. Lists the computations that are executed at the time the page is rendered. Computations are units of logic used to assign session state to items.
- **Processes**. Lists the processes that are executed at the time the page is rendered. Processes are logic controls used to execute data manipulation language (DML) or PL/SQL. For example, you can use a process to populate session state at the time the page is rendered.

The following list describes Page Processing components:

- **Computations**. Lists the computations that are executed at the time the page is processed. Computations are units of logic used to assign session state to items.
- **Validations**. Enables you to create logic controls to verify whether user input is valid. For example, a validation can check whether a value has been typed into a mandatory field.
- **Processes**. Lists the processes that are executed after the page is submitted. Processes are logic controls used to execute data manipulation language (DML) or PL/SQL.
- **Branching**. Enables you to create logic controls that determine how the user navigates through the application.

See Also:

- "Viewing Page Attributes" on page 7-37 and "Editing a Page Definition" on page 7-38 for more information on viewing and editing Page Rendering and Page Processing components
- "Understanding Shared Components" on page 6-7

How Application Builder Uses Templates

The HTML DB engine constructs the look and feel of each page using templates. Templates contains contain HTML and variables that are substituted with dynamic values at runtime. Using templates, the HTML DB engine dynamically renders pages from data stored in tables. You can specify a template for each page. If you do not specify a template, the HTML DB engine uses the default application level template. Application Builder also includes templates for regions, rows, lists, reports, labels, menus, list of values, and buttons.

The use of templates offers a number of advantages:

- Templates can be shared by multiple components
- A single change to a template will affect all components using that template at once

The separation of the user interface definition from data access and application logic ensures that the application and user interface can be built concurrently by different people. The following sections describe the different types of templates available in Oracle HTML DB.

See Also:

- "Working with Templates" on page 7-16
- "About User Interface Templates" on page 7-34
- "About Template Defaults" on page 7-35

Page Templates

Each page is rendered using a page template. Every application has a default page template which applies to all pages in the application, unless you specify a different template.

Page templates control the appearance of navigation bars, parent tabs, and standard tabs. To change an application's default page template, select **Edit Attributes** from the Application Builder home page. An asterisk (*) indicates that a template in another application subscribes to (or is using) this template. Page templates are divided into component areas. Each area supports specific substitution strings.

See Also: "Using Substitution Strings" on page 6-22

Region Templates

Region templates control the appearance of regions. For example, you could use a region template to place a box around some content. Region templates can also be used to control the placement of buttons and region titles. An asterisk (*) indicates that a template in another application subscribes to (or is using) this template.

List Templates

List templates control the appearance of lists. For example, you could create a list to add a list of icons on a home page or third level tabs, or to include a progress indicator. An asterisk (*) indicates that a template in another application subscribes to (or is using) this template.

Report Templates

Report templates control the format of database queries. These templates format the results of reports. An asterisk (*) indicates that a template in another application subscribes to (or is using) this template.

Label Templates

Label templates control the appearance of item labels. For example, you could use label templates to determine how required field labels display. An asterisk (*) indicates that a template in another application subscribes to (or is using) this template.

Menu Templates

Menu templates control the display of menus. You select a menu template when you create a region. For example, menu templates can be used to create breadcrumb style navigation links. An asterisk (*) indicates that a template in another application subscribes to (or is using) this template.

Popup List of Values Templates

Popup List of Values templates control the appearance of popup list of values (or items having the type POPUP LOV). You can create as many popup LOVs as you want, but you can only specify one popup LOV template for each application.

How Page Processing and Page Rendering Work

The HTML DB engine dynamically renders and process pages based on data stored in database tables. To view a rendered version of your application, you run or

submit it to the HTML DB engine. When you run an application, the HTML DB engine relies on two processes:

- **Show Page** is the page rendering process. It assembles all the page attributes (including regions, items, and buttons) into a viewable HTML page.
- **Accept Page** takes care of page processing. It performs any computations, validations, processes, and branching.

When you call a page using a URL, the HTML DB engine is running the Show page or page rendering process. When you submit a page, the HTML DB engine saves the submitted values in the session cache and then performs any computations, validations, or processes.

Understanding Shared Components

Shared components are common components that display on every page within an application. Examples of shared components include:

- Parent and standard tabs
- Navigation bars
- List of values
- Menus
- Lists
- Templates

About Standard Tabs and Parent Tabs

Application Builder includes two different types of tabs:

- Standard tabs
- Parent tabs

An application having only one level of tabs uses **standard tabs**. An application having two levels of tabs uses a parent and standard tabs. A standard tab set is associated with a specific page and page ID. You can use standard tabs to link users to a specific page.

Parent tabs give users another level of navigation. Parent tabs function as a container to hold a group of standard tabs. You can use parent tabs to link users to a specific URL associated with a specific page. When the target page appears, it displays its own standard tab set.

> **See Also:** "Creating Tab Sets" on page 8-2 for more information on creating standard and parent tabs

About Navigation Bars

Use navigation bars to link users to various pages within an application. Typically navigation bars are used to enable users to log in and log out or link to help text. The location of a navigation bar depends upon the associated page template. A navigation bar icon enables you to display a link from an image or text. When you create a navigation bar icon you can specify an image name, text, a display sequence, and a target location (an URL or page).

> **See Also:** "Creating a Navigation Bar" on page 8-4 for more information on creating navigation bars

About List of Values

A list of values (LOV) is a static or dynamic definition used to display a specific type of page item, such as a radio group, check box, or select list. LOVs can be static (that is, based on a set of predefined display and return values) or dynamic (based on SQL queries that select values from tables).

You define LOVs at the application level by running the LOV Wizard and adding them to the Named List of Values repository.

> **See Also:** "Creating LOVs" on page 8-21

About Menus

A menu is a hierarchical list of links that is rendered using a template. For example, you can display menus as a list of links or as a breadcrumb path.

> **See Also:** "Creating Menus" on page 8-7 for more information on creating menus

About Lists

A list is a collection of links that is rendered using a template. For each list entry, you specify display text, a target URL, and other attributes that can control when and how the list entry displays. You control the display of the list by linking it to a template. This template controls the appearance of all list entries.

> **See Also:** "Creating Lists" on page 8-11 for more information on creating lists

About Templates

Templates control the look and feel of the pages in your application. As you create your application you specify templates for pages, regions, reports, lists, labels, menus, buttons, and popup list of values.

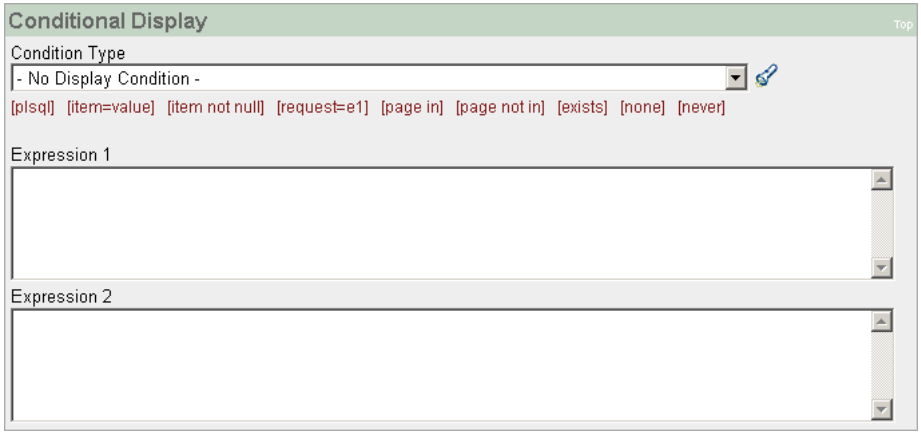
> **See Also:** "Working with Templates" on page 7-16 for more information on viewing, creating, and editing templates

Understanding Conditional Rendering and Processing

A condition is a small unit of logic that helps you control the display of regions, items, buttons, and tabs as well the execution of processes, computations, and validations. For example, when you apply a condition to a button, the rendering engine evaluates the condition during the rendering (or Show page) process. Whether the condition passes or fails determines whether the component (such as a button) displays.

You specify a condition by selecting a condition type when you create the component (that is, the region, item, button, or tab) or by making a selection from the component's conditional display attribute. (See Figure 6–3 on page 6-9.) The condition evaluates to true or false based on the values you enter in the Expression fields.

Figure 6–3 Conditional Display Attribute



To view a complete listing of all available conditions for a given component, click the view icon to the right of the Conditional Display Type list. Shortcuts to common

selections appear directly beneath the Type list. If your condition requires an expression, type it in the appropriate field.

The sections that follow offer examples of some commonly used condition types.

Current Page In Expression 1

`Current page = Expression 1` evaluates to true if the current page matches the page ID listed in the Expression 1 field. For example:

```
100,101,102
```

If the current page is 100, 101, or 102, then this condition evaluates to true and the condition passes.

Exists

`Exists (SQL query returns at least one row)` is expressed as a SQL query. If the query returns at least one row, then the condition evaluates as true. For example:

```
SELECT 1 FROM emp WHERE deptno = :P101_DEPTNO
```

This example references item `P101_DEPTNO` as a bind variable. You can use bind variables within application processes and SQL query regions to reference item session state. If one or more employees are in the department identified by the value of `P101_DEPTNO`, then the condition evaluates as true.

> **See Also:** "About Bind Variables" on page 6-18 for more information

PLSQL Expression

Use `PLSQL Expression` to specify an expression in valid PL/SQL syntax that evaluates to true or false. For example:

```
NVL(:MY_ITEM,'NO') = 'YES'
```

If the value of `:MY_ITEM` is YES, then the condition evaluates as true. Otherwise it evaluates as false.

> **See Also:** Appendix A, "Available Conditions" on page A-1 for a detailed listing of all conditional types available in Oracle HTML DB

Using Build Options to Control Configuration

Build options enable you to conditionally display specific functionality within an application. Using build options you can control which features of an application are turned on for each application deployment. If you specify a build option at the application level, you do not need to specify it for each attribute (for example, each page, branch, button, item, or tab).

Build options have two possible values: INCLUDE and EXCLUDE. If you specify an attribute as being included, then the HTML DB engine considers it part of the application definition at runtime. Reversely, if you specify an attribute as being excluded then the HTML DB engine treats it as if it does not exist.

> **See Also:** "Editing Application Attributes" on page 7-29 and "Editing Page Attributes" on page 7-52 for more information on specifying build options

Creating Build Options

Before you can specify a build option, you must create it.

To create a build option:

1. Click the **Build** icon.
2. When Application Builder appears, select the **Builds** tab.
3. To create a new build option, click **Create**.
4. Follow the on-screen instructions.

Viewing Build Option Reports

Oracle HTML DB includes a report detailing build option utilization in the current application.

To view a report of build option utilization:

1. Click the **Build** icon.
2. When Application Builder appears, select the **Builds** tab.
3. Click **Navigate** in the left navigation pane and then select **Build Opt Utilization**.
4. Make a selection from the Build Options list and click **Go**.

Verifying User Identity

Authentication is the process of establishing users' identities before they can access an application. Authentication may require a user enter a username and password or may involve the use of a digital certificate or a secure key.

Oracle HTML DB supports authentication. You can establish a user's identity by selecting from a number of built-in authentication methods, or by using a wizard to create your own custom authentication approach.

> **See Also:** "Establishing User Identity Through Authentication" on page 10-9 for more information

Controlling Access to Components

While conditions control the rendering and processing of specific components on a page, authorizations control user access to specific components. Authorization is a broad term for controlling access to resources based on predefined user privileges.

Authorization schemes extend the security of your application's authentication scheme. You can specify an authorization scheme for an entire application, a page, or specific component such as a region, item, or button. For example, you could use an authorization scheme to selectively determine which tabs, regions, or navigations bars a users sees.

> **See Also:** "Providing Security Through Authorization" on page 10-17

Understanding Session State Management

HTTP, the protocol over which HTML pages are most often delivered, is a stateless protocol. A Web browser is only connected to the server for as long as it takes to download a complete page. In addition, each page request is treated by the server as an independent event, unrelated to any page requests that have happened previously or may occur in the future. This means that to access form values entered on one page on a subsequent page, some form of session state management needs to occur. Typically, when a user enters values into a form on one page, those values are not accessible on later pages. Oracle HTML DB transparently maintains session state and provides developers with the ability to get and set session state values from any page in the application.

A session is a logical construct that establishes persistence (or stateful behavior) across page views. Each session is assigned a unique identifier within the Oracle HTML DB installation. The HTML DB engine uses this identifier (or session ID) to

store and retrieve an application's working set of data (or session state) before and after each page view.

Because sessions are entirely independent of one another, any number of sessions can exist in the database at the same time. Since sessions persist in the database until purged by an administrator, a user can return to an old session and continue running an application long after first launching it. A user can also run multiple instances of an application simultaneously in different browser sessions.

Oracle HTML DB sessions are logically and physically distinct from the Oracle database sessions used to service page requests. A user runs an application in a single Oracle HTML DB session from log in to log out with a typical duration measured in minutes or hours. Each page requested during that session results in the HTML DB engine creating or reusing an Oracle database session to access database resources. Each of these sessions lasts just a fraction of a second.

Understanding Session IDs

The HTML DB engine establishes the identity (or anonymity) of the user for each page request and the session ID in order to fetch session state from the database. The most visible location of the session ID is in the URL for a page request. Another visible location is in the page's HTML POST structures or in a session cookie sent by the HTML DB engine during authentication and maintained for the life of the application (or browser) session.

Oracle HTML DB assigns new session IDs during authentication processing, records the authenticated user's identity with the session ID, and continually checks the session ID in each page request's URL or POST data with the session cookie and the session record in the database. These checks provide users with both flexibility and security.

While the session ID is the key to session state, the session cookie (where applicable) and the session record safeguard the integrity of the session ID and the authentication status of the user.

Viewing Session State

The behavior of an HTML DB application is usually driven by values in session state. For example, a button may display conditionally based on the value of an item session state. You can view the session state for a page by clicking **Session** on the Developer toolbar.

Figure 6–4 Developer Toolbar

The Session State page provides valuable information about the page. Table 6–1 describes the various types of information available on the Session State page.

Table 6–1 Information Available on the Session State Page

Heading	Description
Application, Page, Session	Identifies the application name, page ID, and session ID.
Page Items	Identify attributes of the page, including the item name, how the item displays (hidden, popup, button, display only HTML), the state or session ID, and status. The Status column indicates the status of the session state. Available values include: ■ I - Inserted ■ U - Updated ■ R - Reset
Application Items	Application items are items that do not reside on a page. Application items are session state variables without the associated user interface properties. **See Also:** "Using Substitution Strings" on page 7-38 for more information on referencing item values
Application Environment	Identifies the session ID, current user, security ID, and browser language.
Session State	Indicates the user's entire session state. The section at the top indicates the state for the current page.

See Also: "Using the Developer Toolbar" on page 7-10 for more information about the Developer toolbar

Managing Session State Values

When building interactive, data driven Web applications, the ability to access and manage session state values with ease is critical. In Oracle HTML DB, session state is automatically managed for every page and easily referenced in static HTML or logic controls such as processes or validations.

Topics in this section include:

- Referencing Session State
- Setting Session State
- Clearing Session State
- About Bind Variables

> **See Also:** "About Items" on page 7-43 and "Referencing Item Values" on page 7-47 for more information

Referencing Session State

Referencing the value of an item is one of the most common examples of referencing session state. In Oracle HTML DB, an item can be a field, a text area, a password, a select list, or checkbox. Table 6–2 describes the supported syntax for referencing *item values*.

Table 6–2 Syntax for Referencing Item Values

Type	Syntax	Description
SQL	`:MY_ITEM`	Standard bind variable syntax for items no longer than 30 bytes. Use this syntax for references within a SQL query and within PL/SQL.
PL/SQL	`v('MY_ITEM')`	PL/SQL syntax referencing the item value using the v function. **See Also:** "Oracle HTML DB APIs" on page 13-1
PL/SQL	`nv('MY_NUMERIC_ITEM')`	Standard PL/SQL syntax referencing the numeric item value using the nv function. **See Also:** "Oracle HTML DB APIs" on page 13-1
Static Text	`&MY_IITEM`	Static text.
Static Text (exact)	`&MY_IITEM.`	Static text. Exact Substitution.

Setting Session State

When a user submits a page in Oracle HTML DB, the HTML DB engine automatically stores values typed into fields (items) in session state. For example, suppose you have an application containing two pages. The first page of the application contains a form in which a user can enter a phone number. You have

defined this form by creating an item named *P2_PhoneNo*. On the second page you want to display the information the user enters in the form.

When the page is submitted, Oracle HTML DB captures the value typed in the phone number field and stores the value for future use. The phone number typed by the user can then be retrieved from session state by referencing the item associated with the field on the page.

Clearing Session State

As you develop your applications, you may find it useful to clear the cached value for specific items, all items on a page, all pages in an application, or the current user session. Clearing a cached value resets the value to null. The topics that follow offer specific examples of clearing session state.

Clearing Cache by Item

Clearing cache for a single item resets the value of the item to null. For example, you might use this approach to make sure a specific item's value is null when a page is prepared for rendering.

The following example uses standard `f?p` syntax to clear the cache for an item. This example calls page 5 of application 100. Placing `MY_ITEM` in the `ClearCache` position of the `f?p` syntax resets the value of `MY_ITEM` to `NULL`.

```
f?p=100:5:&SESSION.::NO::MY_ITEM:
```

The following example resets the value of the items *THE_EMPNO* and *THE_DEPTNO*.

```
f?p=100:5:&SESSION.::NO::THE_EMPNO,THE_DEPTNO:,
```

Clearing Cache by Page

Caching application items provides a very effective way to maintain session state. However, there are occasions when you may want to clear the cache for all items on a page. For example, suppose you needed to clear all fields on page when a user clicks a link the creates a new order. By clearing cache for an entire page you set the value of all items on the page to null.

Clearing Session Cache for Two Pages While Resetting Pagination This example clears the session cache for two pages and resets pagination.

```
f?p=6000:6003:&SESSION.::NO:RP,6004,6014
```

This example:

- Runs page 6003 of application 6000 and uses the current session ID
- Indicates to not show debug information (NO)
- Clears all values maintained by the current session's cache for items of pages 6004 and 6014
- Resets region pagination (RP) on page 6003 (the requested page)

See Also: Creating a Report with Pagination on page 8-35

Clearing Session Cache on a Page and Passing an Item Value This example demonstrates a good way to implement an update form. It clears existing information and sets the item's value (typically a primary key).

```
f?p=6000:6003:&SESSION.::NO:6003:MY_ITEM:1234
```

Specifically this example:

- Runs page 6003 of application 6000 and use the current session ID
- Indicates to not show debug information (`NO`)
- Clears all values maintained by the current session's cache for items on page 6003
- Sets the session state of an item called `MY_ITEM` to the value `1234`

Clearing Session Cache on a Page and Passing Values to Multiple Items This example is similar to the previous one except it passes values to multiple items.

```
f?p=6000:6004:&SESSION.::NO:6003:MY_ITEM1,MY_ITEM2,MY_ITEM3:1234,,5678
```

Specifically this example:

- Runs page 6004 of application 6000 and use the current session ID
- Clears the current session's cache for items on page 6003
- Indicates debug information should be hidden (`NO`)
- Sets the value of `MY_ITEM1` to 1234, sets the value of `MY_ITEM2` to null (indicated by the commas used as placeholder), and set the value of `MY_ITEM3` to 5678

Clearing Cache for an Entire Application

You can also clear application cache by using `f?p` syntax by creating a `REQUEST` argument using the keyword *APP* using the following syntax:

```
f?p=App:Page:Session::NO:APP
```

Note: Resetting the cache for an entire application does not actually restore the application to a completely reset state. For example, if an application includes on-new instance computations or on-new instance processes, the HTML DB engine runs these computations and processes when the application session is created. Then, it processes the clear cache request and displays the requested page.

The only way to reset the application completely is to request it using a URL without a session ID, or by calling `HTMLDB_APPLICATION.CLEAR_APP_CACHE` from another application. If the session ID is set using a cookie, you will need to logout in order to reset the state.

Clearing Cache for the Current User Session

You can also clear application cache by using `f?p` syntax. Create a `REQUEST` argument using the keyword `SESSION`. For example:

```
f?p=6000:6004:12507785108488427528::NO:SESSION
```

About Bind Variables

You can use bind variables within an application process or SQL query to reference session state of a specified item. For example:

```
SELECT * FROM emp WHERE name like '%' || :SEARCH_STRING || '%'
```

In this example, the search string is a page item. If the region type is defined as SQL Query, you can reference the value using standard SQL bind variable syntax. Using bind variables ensures that parsed representations of SQL queries are reused by the database, optimizing memory usage by the server.

When using bind variable syntax remember the following rules:

- Bind variable names must correspond to an item name

- Bind variable names are not case sensitive
- Bind variable names cannot be longer than 30 characters (that is, they must be a valid Oracle identifier)

  Although application items can be up to 255 characters, if you intend to use an application items within SQL using bind variable syntax, the item name must be 30 characters or less.

Using Bind Variables in Regions Based on a SQL Query or LOV

If your region type is defined as a SQL Query, SQL Query (plsql function body returning SQL query), or list of values (LOV) you can reference session state using the syntax:

```
:MY_ITEM
```

One common way to do this is to incorporate a session state variable in a WHERE clause. The following example demonstrates how to bind the value of the item `THE_DEPTNO` into a region defined from a SQL Query.

```
SELECT ename, job, sal
FROM emp
WHERE deptno = :THE_DEPTNO
```

> **See Also:** Creating Regions on page 8-14 for more information on creating different types of regions

Using Bind Variables in PL/SQL Procedures

For region types defined as a PL/SQL Procedure, regions are constructed using PL/SQL anonymous block syntax. In other words, the beginning and ending are added automatically around the PL/SQL. For example:

```
INSERT INTO emp (empno, ename, job)
VALUES (:P1_empno, :P1_name, :P1_job);
```

In this example, the values of the `empno`, `ename`, and `job` are populated by the values of `P1_empno`, `P1_name`, and `P1_job`.

Understanding URL Syntax

Each application has a number (called an application ID) or alphanumeric alias which uniquely identifies it. Similarly, each page also has a unique number (called a

page ID) or an alphanumeric alias. When you run an application, the HTML DB engine generates a session number that serves as a key to the user's session state.

The URL that displays for each page indicates the location of Oracle HTML DB and identifies the application ID, page ID, and session ID. For example:

```
http://marvel.oracle.com/pls/otn/f?p=4350:1:220883407765693447
```

This example indicates that:

- The address of Oracle HTML DB is:

  ```
  http://marvel.oracle.com/pls/otn/
  ```

- The application ID is 4350
- The page ID is 1
- The session ID is 220883407765693447

Using f?p Syntax to Link Pages

You can create links between pages in your application using the following syntax:

```
f?p=App:Page:Session:Request:Debug:ClearCache:itemNames:itemValues:PrinterFriendly
```

Table 6–3 describes the possible arguments you can pass when using `f?p` syntax.

Table 6–3 f?p Syntax Arguments

Syntax	Description
`App`	Indicates an application ID or alphanumeric alias.
`Page`	Indicates a page ID or alphanumeric alias.
`Session`	Identifies a session ID. You can reference a session ID to create hypertext links to other pages that maintain the same session state by passing the session number. You can reference the session ID using the syntax: ■ Short substitution string: `&SESSION.` ■ PL/SQL: `v('SESSION')` ■ Bind variable: `:APP_SESSION`

Table 6–3 f?p Syntax Arguments

Syntax	Description
Request	Sets the value of REQUEST. Each application button sets the value of REQUEST to the name of the button. This enables accept processing to reference the name of the button when a user clicks it. You can reference REQUEST using the syntax: ■ Substitution string: &REQUEST. ■ PL/SQL: v('REQUEST') ■ Bind variable: :REQUEST
Debug	Displays application processing details. Valid values for the DEBUG flag are YES or NO. Setting this flag to YES displays details about application processing. You can reference the Debug flag using the following syntax: ■ Short substitution string: &DEBUG. ■ PL/SQL: v('DEBUG') ■ Bind variable: :DEBUG
ClearCache	Clears cache. Clearing cache for a single item simply sets the value of the item to null. To clear cached items, use a comma delimited list of page numbers. Comma delimited lists can also contain collections to be reset or the keyword RP, which resets region pagination on the requested page.
itemNames	Comma delimited list of item names used to set session state with an URL.
itemValues	Comma delimited list of item values used to set session state with an URL.
PrinterFriendly	Identifies the printer friendly preference of YES or NO (default). You can use this item when preparing a page view with minimal graphics and a layout suitable for printing. You can reference the printer friendly preference by using the syntax: v('PRINTER_FRIENDLY') When referenced, the HTML DB engine will not display tabs or navigation bars and all items will be displayed as text and not as form elements.

Although it is important to understand how f?p syntax works, you rarely have to construct this syntax yourself. Oracle HTML DB includes many wizards that automatically create these references for you. The sections that follow describe a number of specific instances that utilize f?p syntax to link pages.

Calling a Page Using an Application and Page Alias

The following example calls a page using an application and a page alias from within an Oracle HTML DB application. It runs the page *home* of the application *myapp* and uses the current session ID.

```
f?p=myapp:home:&SESSION.
```

Calling a Page from a Button URL

When you create a button, you can specify a URL to redirect to when the user clicks the button. This example runs page 6001 of application 6000 and uses the current session ID.

```
f?p=6000:6001:&SESSION.
```

Using Substitution Strings

You can use substitution strings within a page template or region source to replace a character string with another value. As you design your application and enable users to edit items, you will need to use substitution strings in order to pass information.

You can use substitution strings in Oracle HTML DB in the following ways.

- Include a substitution string within a template.
- Create an item at the application or page level.
- Use built-in substitution strings to achieve a specific type of functionality.

Substitution strings used within a template contains the pound symbol (#). For example:

```
#ABC#
```

Substitution strings created at the application level do not display, but are used as variable to maintain session state. You can define page items as an attribute of a page. You can use this type of session state substitution at the application or page level. For example:

```
&MY_ITEM.
```

Built-in Substitution Strings

Application Builder supports a number of built-in substitution strings. You may need to reference these values in order to achieve specific types of functionality. Built-in substitution strings available in Oracle HTML DB include:

The sections that follow describe these substitution strings, when to use them, and what supported syntax is currently available. Note that no short syntax exists for *SESSION* or *USER* since both are Oracle reserved words.

Topics in this section include:

- APP_SESSION
- APP_USER
- IMAGE_PREFIX
- WORKSPACE_IMAGES
- APP_IMAGES
- BROWSER_LANGUAGE
- PRINTER_FRIENDLY
- HOME_LINK
- PROXY SERVER
- REQUEST
- SYSDATE_YYYYMMDD
- DEBUG
- APP_ID
- APP_PAGE_ID
- APP SCHEMA OWNER
- SQLERRM
- AUTHENTICATED_URL_PREFIX
- LOGOUT_URL
- PUBLIC_URL_PREFIX
- CURRENT_PARENT_TAB_TEXT
- APP_ALIAS

- APP_UNIQUE_PAGE_ID

See Also:

- "About Static Substitution Strings" on page 7-36 for more information on defining static substitution strings as an application attribute
- "Providing Security Through Authorization" on page 10-17 for more information on authentication

APP_SESSION

APP_SESSION is one of the most commonly used built-in substitution string. You can use this substitution string to create hypertext links between application pages that maintain a session state by passing the session number. Table 6–4 describes the supported syntax for referencing APP_SESSION.

Table 6–4 APP_SESSION Syntax

Reference Type	Syntax
Bind variable	:APP_SESSION
PL/SQL	v('APP_SESSION')
Short PL/SQL	v('SESSION')
Short substitution string	&SESSION.
Substitution string	&APP_SESSION.

Consider the following examples:

- From within an HTML Region:

```
<a href="f?p=100:5:&SESSION.">click me</a>
```

- Using PL/SQL:

```
htf.anchor('f?p=100:5:'||v('SESSION'),'click me');
```

- Using a SQL query:

```
SELECT htf.anchor('f?p=100:5:'||:app_session,'clickme') FROM DUAL;
```

APP_USER

The APP_USER is the current user running the application. Depending upon your authentication model, the value of the user set differently. If the application is running using database authentication, then the value of the user is the same as the database pseudo column USER. Table 6–5 describes the supported syntax for referencing APP_USER.

Table 6–5 APP_USER Syntax

Reference Type	Syntax
Bind variable	:APP_USER
PL/SQL	v('APP_USER')
Short PL/SQL	v('USER')
Substitution string	&USER.

Consider the following examples:

- From within an HTML Region:

```
Hello you are logged in as &USER.
```

- Using PL/SQL:

```
htp.p('Hello you are logged in as'||v('USER'));
```

- As a bind variable:

```
SELECT * FROM some_table WHERE user_id = :app_user
```

IMAGE_PREFIX

The Image Prefix attribute is defined on the Edit Application Attributes page. By default the image prefix is "/i/". Use this prefix when referencing images distributed with Oracle HTML DB. If you wish to reference uploaded images, use WORKSPACE_IMAGES and APP_IMAGES. Table 6–6 describes the supported syntax for referencing IMAGE_PREFIX.

> **See Also:** "Editing Application Attributes" on page 7-29 for more information on application attributes

Table 6–6 IMAGE_PREFIX Syntax

Reference Type	Syntax
Bind variable	`:IMAGE_PREFIX`
Direct PL/SQL	`HTMLDB_APPLICATION.G_IMAGE_PREFIX`
PL/SQL	`v('IMAGE_PREFIX')`
Substitution string	`&IMAGE_PREFIX.`
Template Substitution	`#IMAGE_PREFIX#`

WORKSPACE_IMAGES

The workspace images prefix is derived from the `IMAGE_PREFIX` (see "IMAGE_PREFIX"). Workspace images differ from application images in that workspace images are not specific to a given application and can be shared among multiple applications. Table 6–7 describes the supported syntax for referencing `WORKSPACE_IMAGES`.

Table 6–7 WORKSPACE_IMAGES Syntax

Reference Type	Syntax
Bind variable	`:WORKSPACE_IMAGES`
Direct PL/SQL	Not available.
PL/SQL	`v('WORKSPACE_IMAGES')`
Substitution string	`&WORKSPACE_IMAGES.`
Template Substitution	`#WORKSPACE_IMAGES#`

APP_IMAGES

The application images prefix is derived from the IMAGE_PREFIX (see previous section). Oracle HTML DB stores uploaded images in the database. Workspace images are specific to a given application and are not shared over many applications. If you uploaded an image and make it specific to an application, then you must use this substitution string, or bind prefix. Table 6–8 describes the supported syntax for referencing `APP_IMAGES`.

Table 6–8 APP_IMAGES Syntax

Reference Type	Syntax
Bind variable	`:APP_IMAGES`

Table 6–8 APP_IMAGES Syntax

Reference Type	Syntax
Direct PL/SQL	Not available.
PL/SQL	`v('APP_IMAGES')`
Substitution string	`&APP_IMAGES.`
Template Substitution	`#APP_IMAGES#`

BROWSER_LANGUAGE

`BROWSER_LANGUAGE` refers to the Web browser's current language preference. Table 6–9 describes the supported syntax for referencing `BROWSER_LANGUAGE`.

Table 6–9 BROWSER_LANGUAGE Syntax

Reference Type	Syntax
Bind variable	`:BROWSER_LANGUAGE`
Direct PL/SQL	`HTMLDB_APPLICATION.G_BROWSER_LANGUAGE`
PL/SQL	`v('BROWSER_LANGUAGE')`
Substitution string	`:BROWSER_LANGUAGE.`
Substitution string	`&BROWSER_LANGUAGE.`

PRINTER_FRIENDLY

The value of `PRINTER_FRIENDLY` determines whether the HTML DB engine is running in print view mode. This setting can be referenced in conditions to eliminate elements not desired in a printed document from a page. Table 6–10 describes the supported syntax for referencing `PRINTER_FRIENDLY`.

Table 6–10 PRINTER_FRIENDLY Syntax

Reference Type	Syntax
Direct PL/SQL	`HTMLDB_APPLICATION.G_PRINTER_FRIENDLY (VARCHAR2 DATATYPE)`
PL/SQL	`v('PRINTER_FRIENDLY')`
Substitution string	`&PRINTER_FRIENDLY.`

HOME_LINK

HOME_LINK is the home page of an application. The HTML DB engine will redirect to location if no page is given. Table 6–11 describes the supported syntax for referencing HOME_LINK.

Table 6–11 HOME_LINK Syntax

Reference Type	Syntax
Direct PL/SQL	HTMLDB_APPLICATION.G_HOME_LINK
PL/SQL	v('HOME_LINK')
Template Reference	#HOME_LINK#
Substitution String	&HOME_LINK.

PROXY SERVER

PROXY SERVER is an application attribute. The attribute may be used by regions whose source comes from a URL. The following is the correct syntax for a direct PL/SQL reference used when you are writing PL/SQL to access remote Web servers from within the database (for example, when using the utl_http package shipped with the database).

```
HTMLDB_APPLICATION.G_PROXY_SERVER
```

REQUEST

Each application button sets the value of REQUEST to the name of the button. This enables accept processing to reference the name of the button when a user clicks it. In the f?p syntax, REQUEST may be set using the fourth argument.

Referencing the Value of REQUEST REQUEST is typically referenced during Accept processing (that is, the processing that occurs when you post a page). Table 6–12 describes the supported syntax for referencing REQUEST.

Table 6–12 REQUEST Syntax

Reference Type	Syntax
Bind variable	:REQUEST
Direct PL/SQL	HTMLDB_APPLICATION.G_REQUEST
PL/SQL	v('REQUEST')

Table 6–12 REQUEST Syntax

Reference Type	Syntax
Substitution string	`&REQUEST`
	`&REQUEST.` (exact syntax match)

Scope and Value of REQUEST for Posted Pages When you post a page, you initiate Accept processing. Accept processing consists of computations, validations, processes, and branches. The value of request is available during each phase of the Accept processing. Once Oracle HTML DB branches to a different page then `REQUEST` is set to NULL.

The value of `REQUEST` is the name of the button the user clicks, or the name of the tab the user selects. For example, suppose you have a button with a name of "CHANGE", and a label of "Apply Change." When a user clicks the button the value of `REQUEST` will be `CHANGE`.

Referencing REQUEST Using Declarative Conditions It is common to reference `REQUEST` using conditions. For example, you may wish to reset pagination when a user clicks **Go** on a report page. You can reset pagination by creating a on-submit page process. The page process can be made conditional using the condition `Request = Expression 1`.

To create an on-submit page process:

1. Under Conditional Display, select the condition type **Request = Expression 1**.
2. In Expression 1, enter **GO**.

Using REQUEST for Show Processing You can also use `REQUEST` for Show processing when navigating to a page using `f?p` syntax. For example:

```
f?p=100:1:&SESSION.:GO
```

Remember that the fourth argument is in the `f?p` syntax is `REQUEST`. This example goes to application 100, page 1 for the current session and sets the value of `REQUEST` to GO. Any process or region can reference the value of `REQUEST` using Show processing.

The following is a similar example using PL/SQL:

```
IF v ('REQUEST') = 'GO' THEN
   htp.p('hello');
END IF;
```

Note that `htp.p('hello')` is a call to a PL/SQL Web Toolkit package in order to print out the specified text string.

See Also:

- *Oracle Database Application Developer's Guide - Fundamentals* for more information on developing Web applications with PL/SQL
- *PL/SQL Packages and Types Reference* for more information on htp packages

SYSDATE_YYYYMMDD

`SYSDATE_YYYYMMDD` represents the current date on the database server, with the *`YYYYMMDD`* format mask applied. You may use this value instead of repeated calls to the SYSDATE() function. The following list describes the supported syntax for referencing `SYSDATE_YYYYMMDD`.

- Bind variable:

  ```
  :SYSDATE_YYYYMMDD
  ```

- PL/SQL:

  ```
  v('SYSDATE_YYYYMMDD')
  ```

- Direct PL/SQL:

  ```
  HTMLDB_APPLICATION.G_SYSDATE (DATE DATATYPE)
  ```

Table 6–13 SYSDATE_YYYYMMDD Syntax

Reference Type	Syntax
Bind variable	`:SYSDATE_YYYYMMDD`
Direct PL/SQL	`HTMLDB_APPLICATION.G_SYSDATE (DATE DATATYPE)`
PL/SQL	`v('SYSDATE_YYYYMMDD')`

DEBUG

Valid values for the `DEBUG` flag are YES or NO. Turning debug on shows details about application processing. If you write your own custom code, you may wish to

generate debug information only if the debug mode is set to YES. Table 6–14 describes the supported syntax for referencing DEBUG.

Table 6–14 DEBUG Syntax

Reference Type	Syntax
Bind variable	:DEBUG
Direct PL/SQL	HTMLDB_APPLICATION.G_DEBUG
PL/SQL	v('DEBUG')
Substitution string	&DEBUG.

The following is an example of a substitution string reference that preserves the current value of DEBUG:

```
f?p=100:1:&SESSION.::&DEBUG
```

APP_ID

APP_ID identifies the application ID of the currently executing application. Table 6–15 describes the supported syntax for referencing APP_ID.

Table 6–15 APP_ID Syntax

Reference Type	Syntax
Bind variable	:APP_ID
Direct PL/SQL	HTMLDB_APPLICATION.G_FLOW_ID (A NUMBER)
PL/SQL	nv('APP_ID')
Substitution string	&APP_ID.

The following is an example of a substitution string reference:

```
f?p=&APP_ID.:40:&SESSION.
```

APP_PAGE_ID

APP_PAGE_ID is the current application Page ID. For example, if your application was on page 3 then the result would be 3. Using this syntax is useful when writing

application components that need to work generically in multiple applications. Table 6–16 describes the supported syntax for referencing APP_PAGE_ID.

Table 6–16 APP_PAGE_ID Syntax

Reference Type	Syntax
Bind variable	:APP_PAGE_ID
Direct PL/SQL	HTMLDB_APPLICATION.G_FLOW_STEP_ID (A NUMBER)
Direct PL/SQL	:APP_PAGE_ID
PL/SQL	nv('APP_PAGE_ID')
Substitution string	&APP_PAGE_ID.

The following is an example of a substitution string reference:

```
f?p=&APP_ID.:&APP_PAGE_ID.:&SESSION.
```

APP SCHEMA OWNER

If you are generating calls to applications from within your PL/SQL code, you may need to reference the owner of the Oracle application schema. The following describes the correct syntax for direct PL/SQL reference:

```
HTMLDB_APPLICATION.G_FLOW_SCHEMA_OWNER
```

SQLERRM

SQLERRM is template substitution only available in the Applications Region Error Message. The following describes the correct syntax for a region template substitution reference:

```
#SQLERRM#
```

AUTHENTICATED_URL_PREFIX

This application level attribute identifies a valid authenticated prefix (that is, a logged in URL prefix). You can use a relative path or a full path beginning with http. This item is useful if your application can be run in both authenticated (logged in) and public (not logged in) modes. You can use AUTHENTICATED_URL_PREFIX to construct a link to an authenticated page. This item is most useful when using basic database authentication since changes to the URL can require

authentication. Table 6–17 describes the supported syntax for referencing AUTHENTICATED_URL_PREFIX.

Table 6–17 AUTHENTICATED_URL_PREFIX Syntax

Reference Type	Syntax
Bind variable	:AUTHENTICATED_URL_PREFIX
PL/SQL	v('AUTHENTICATED_URL_PREFIX')
Substitution string	&AUTHENTICATED_URL_PREFIX.

LOGOUT_URL

LOGOUT_URL is application level attribute used to identify the logout URL. This is a URL that navigates the user to a logout page or optionally directly logs a user out. To create a logout navigation bar icon, use &LOGOUT_URL for the navigation bar link. If you are coding a page template use #LOGOUT_URL#. Table 6–18 describes the supported syntax for referencing LOGOUT_URL.

Table 6–18 LOGOUT_URL Syntax

Reference Type	Syntax
Bind variable	:LOGOUT_URL
PL/SQL	V('LOGOUT_URL')
Substitution string	&LOGOUT_URL.
Template Substitution	#LOGOUT_URL#

PUBLIC_URL_PREFIX

PUBLIC_URL_PREFIX is an application level attribute that identifies an URL to toggle out of a logged in mode to a public view. Table 6–19 describes the supported syntax for referencing PUBLIC_URL_PREFIX.

Table 6–19 PUBLIC_URL_PREFIX Syntax

Reference Type	Syntax
Bind variable	:PUBLIC_URL_PREFIX
PL/SQL	v('PUBLIC_URL_PREFIX')
Substitution string	&PUBLIC_URL_PREFIX.
Template Substitution	#PUBLIC_URL_PREFIX#

CURRENT_PARENT_TAB_TEXT

CURRENT_PARENT_TAB_TEXT is most useful in page templates, but is only relevant for applications that use two level tabs (that is, parent and standard tabs). Use this string to reference the parent tab label. This substitution string enables you to repeat the currently selected parent tab within the page template. Table 6–20 describes the supported syntax for referencing CURRENT_PARENT_TAB_TEXT.

Table 6–20 CURRENT_PARENT_TAB_TEXT Syntax

Reference Type	Syntax
Bind variable	Not Available.
Substitution string	&CURRENT_PARENT_TAB_TEXT.

APP_ALIAS

APP_ALIAS is alphanumeric name for the current application. APP_ALIAS is different from the APP_ID in that the APP_ID must be unique over all companies and all applications hosted in one database. In contrast, APP_ALIAS must be unique within a workspace. Using the same APP_ALIAS you can create an application called ABC in two workspaces. You can use APP_ALIAS almost anywhere a APP_ID can be used. For example, f?p Syntax can use an APP_ALIAS or an application ID as demonstrated in this example:

```
f?p=ABC:1:&SESSION.
```

This example runs application ABC, page 1 using the current session.

Table 6–21 describes the supported syntax for referencing APP_ALIAS.

Table 6–21 APP_ALIAS Syntax

Reference Type	Syntax
Bind variable	:APP_ALIAS
PL/SQL	v('APP_ALIAS')
Substitution string	&APP_ALIAS.

The following is an HTML example:

```
Click me to go to page 1 <a href="f?p=&APP_ALIAS.:1:&SESSION."> of the current
application</a>
```

APP_UNIQUE_PAGE_ID

APP_UNIQUE_PAGE_ID is an integer generated from an Oracle sequence which is unique for each page view. This number is used by applications to prevent duplicate page submissions and can be used for other purposes. For example, if you wish to make a unique URL to avoid browser caching issues, you can embed this number in the request or debug column in calls to the f procedure. Table 6–22 describes the supported syntax for referencing APP_UNIQUE_PAGE_ID.

Table 6–22 APP_UNIQUE_PAGE_ID Syntax

Reference Type	Syntax
Bind variable	:APP_UNIQUE_PAGE_ID
PL/SQL	v('APP_UNIQUE_PAGE_ID')
Substitution string	&APP_UNIQUE_PAGE_ID.

The following is an HTML example:

```
SELECT 'f?p=100:1:'||:APP_SESSION||':'||:APP_UNIQUE_PAGE_ID||
    ':::P1_EMPNO:'||empno,
   ename,
    job
FROM emp
```

Note the use of the APP_UNIQUE_PAGE_ID in the request column. This makes this URL unique and may avoid excessive browser caching problems.

7

Using Application Builder

This section provides information on how to use Application Builder. Application Builder is the tool you use to build the pages that comprise an application.

This section contains the following topics:

- Understanding the Definition of a Page
- Creating an Application
- Creating a New Page Using a Wizard
- Working with Templates
- Viewing Application Attributes
- Editing Application Attributes
- Viewing Page Attributes
- Editing a Page Definition
- Running a Page

See Also:

- Chapter 1, "What is Oracle HTML DB?"
- Chapter 2, "Quick Start"
- Chapter 6, "Application Builder Concepts"
- Chapter 8, "Building Application Components"

Understanding the Definition of a Page

You use Application Builder to build dynamically rendered applications in Oracle HTML DB. An application is a collection of database-driven Web pages linked together using tabs, buttons, or hypertext links.

A page is the basic building block of an application. Each page can have buttons and fields (called items) and can include application logic (or processes). You can branch from one page to the next using conditional navigation, perform calculations (called computations), validations (such as edit checks), and display reports, calendars, and charts.

Topics in this section include:

- Accessing Application Builder
- Viewing a Page Definition
- Viewing Page Reports
- Using the Developer Toolbar

Accessing Application Builder

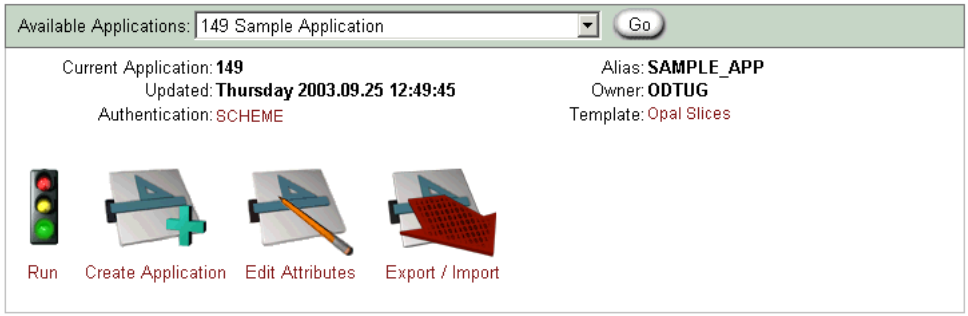
An application is a collection of pages that share a common session state definition and authentication method. Application Builder is the tool you use to build the pages that comprise an application.

To access the Application Builder home page, navigate to the Development tab and click **Application Builder**. The Application Builder home page appears. The sections that follow describe the Application Builder home page.

About the Available Applications List

Use the Available Applications list (see Figure 7–1 on page 7-3) to select an existing application and then click **Go**.

Figure 7–1 Available Applications List



The current application ID, last update date, application authentication scheme, the application alias, the application owner, and selected template display directly beneath the list. You can run an existing application, create a new application, edit application attributes, or export information by clicking the following icons:

- **Run** submits the pages in the current application to the HTML DB engine to render viewable HTML.
- **Create Application** creates a new application using the Create Application Wizard.
- **Edit Attributes** displays the Edit Application Attributes page.
- **Export/Install** links you to the Export Import Wizard.

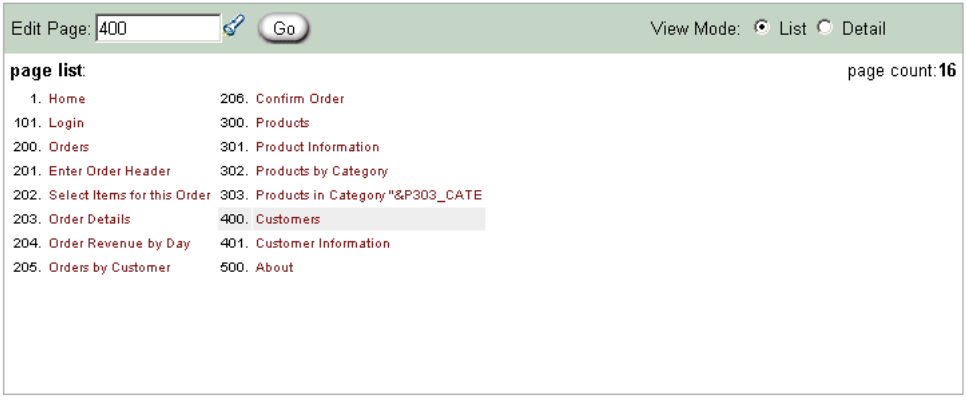
See Also:

- "Running a Page" on page 7-57
- "Creating a New Page Using a Wizard" on page 7-13
- "Editing Application Attributes" on page 7-29
- "Exporting and Importing Applications" on page 10-2

About the Edit Page List

As shown in Figure 7–2 on page 7-4, a page list displays at the bottom of the page. To access a specific page, enter a page ID in the Edit Page field and click **Go**. To change the view mode, select **List** or **Detail**.

Figure 7–2 Page List

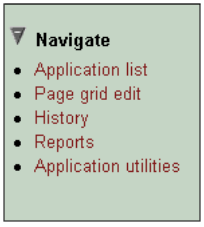


List view displays only the page ID and page name. To edit a page, drill down on the page name. Detail view displays a report of all pages that comprise your application. To sort, click the column headings. The report displays the page ID, name, and counts of all components on the page.

About the Application Navigation Pane

The Application Navigation pane displays on the left side of the page. Click **Navigate** to expand or collapse the list.

Figure 7–3 Application Navigation Pane



As shown in Figure 7–3 on page 7-4, the available links include:

- **Application list** displays all applications in your workspace to which you have edit access. Change the view by selecting List or Detail at the top of the page. To edit an application in Detail view, click the edit icon. To edit an application in List view, select the application name.

- **Page grid edit** displays an editable table of all pages in the currently selected application. To apply your edits, click Apply Changes. To create a new page using the Create Page Wizard, click **Create New**. To edit a page, click the edit icon.
- **History** displays counts of changes to applications aggregated by user by day.
- **Reports** displays links to summary application reports. Summary Reports offer interesting summary information such as details about customized regions, component counts, and related pages in the current application. Utilization Reports offer information specific to how components are utilized on each page.
- **Application Utilities** offers quick access utilities that enable you to manage translations, cascading style sheets, images, and export and import applications.

Viewing a Page Definition

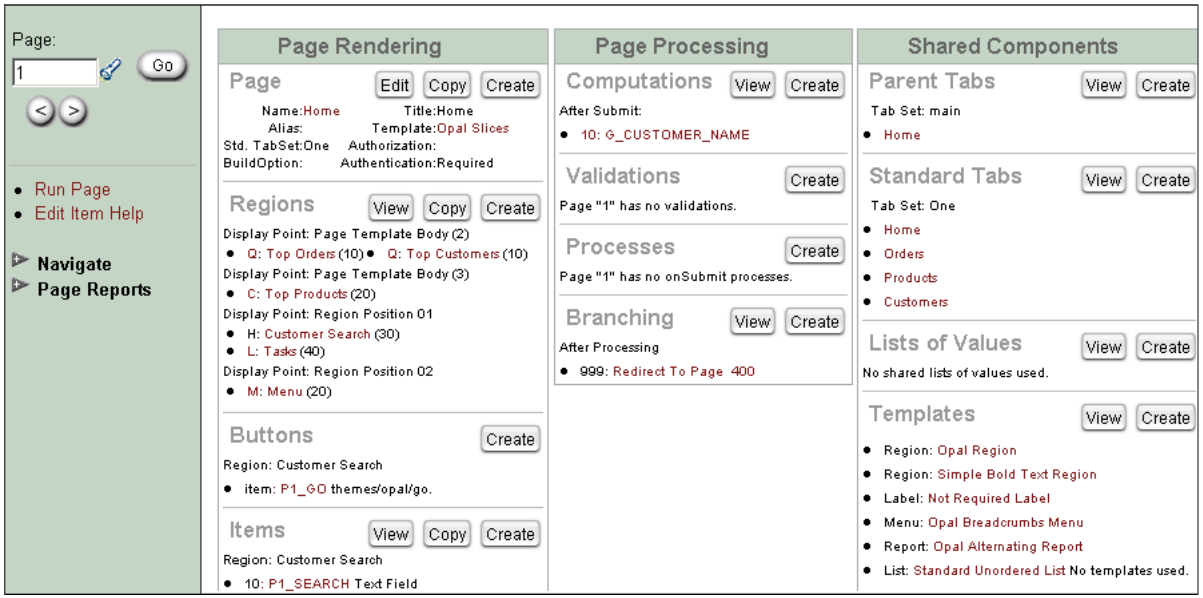
You can view, create, and edit the components that define a page by accessing a Page Definition.

To view the Page Definition for an existing page:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. To edit a specific page, you can either:
   - Enter the page ID in the Edit Page field and click **Go**
   - Click the page name

Figure 7–4 Page Definition



As shown in Figure 7–4, the Page Definition appears.

Every Page Definition is divided into four sections as described in Table 7–1.

Table 7–1 Page Definition Page

Section	Description
Navigation Pane	Enables you to link to another page, run the current page, edit item help as well as display navigation links and page reports. **See Also:** "Using the Page Navigation Pane" on page 7-7 for more information
Page Rendering	Defines all attributes for the page, regions, buttons, items, page rendering computations and page level processes. **See Also:** "Viewing Page Attributes" on page 7-37 and "Managing Page Rendering Components" on page 7-39 for more information
Page Processing	Specifies page level application logic such as computations, validations, processes, and branching. **See Also:** "About Page Processing Components" on page 7-50 for more information

Table 7–1 Page Definition Page

Section	Description
Shared Components	Displays application components that can be shared among multiple pages. **See Also:** "Creating Tab Sets" on page 8-2, "Creating a Navigation Bar" on page 8-4, "Creating Menus" on page 8-7, "Creating Lists" on page 8-11, and "Editing Templates" on page 7-17

Using the Page Navigation Pane

The Page Navigation pane displays on the left side of the page. Available options include:

- **Page.** Use this field to link to other pages. To access a page directly, enter a page ID and click **Go**. To access the previous or next page ID, click the arrow buttons.
- **Run Page.** Submits the pages in the current application to the HTML DB engine. The HTML DB engine then renders the pages into viewable HTML.
- **Edit Item Help.** Enables you to edit help text associated with all items on the current page. Item help text serves as context sensitive help for users of an application.
- **Navigate.** Expandable list offering quick access to wizards for creating lists, lists of values, menus, security, tabs, and templates. Click **Navigate** to expand or collapse the list.
- **Page Reports.** Expandable list offering access to page reports. Click **Page Reports** to expand or collapse the list.

  See Also:

  - "Creating a Help Page" on page 8-41 for more information on creating item help
  - "Creating Reports" on page 8-28 for more information on page reports

Viewing Page Reports

Every Page Definition includes a Page Reports list in the left navigation pane. Each report offers a different presentation of the components that define the page.

To access the Page Reports link:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. To access the Page Definition for a specific page, you can either:
   - Enter the page ID in the Edit Page field and click **Go**
   - Click the page name
4. Click **Page Reports** in the left navigation pane to expand the list.
5. Click the appropriate page report link.

The sections that follow describe available page reports.

About All Conditions

Clicking **All Conditions** displays the Conditions Report. This report describes all currently defined conditions on the current page for regions, buttons, items, computations, validations, processes, and branches.

A condition is a small unit of logic that enables you to control the display of regions, items, buttons, and tabs and the execution of processes, computations, and validations. When you apply a condition to a component, the condition is evaluated. Whether the condition passes or fails determines whether the component displays, or whether a logic control (process, computation, validation) is executed.

> **See Also:** "Understanding Conditional Rendering and Processing" on page 6-9

About Event View

Clicking **Event View** displays the Page Event View report. This report details all currently defined page components and processes. It provides a chronological view of how and in what order the HTML DB engine renders the page, invokes logic, and runs processes. A graphical legend in the left navigation pane identifies the component type. You can choose from two view options:

- **Show All** displays all possible page components and processes, including those not currently defined.
- **Show Used** displays currently used page components and processes (Default).

To view details about a specific page component or process, click the appropriate hypertext link.

To run the current page, click **Run**. To create a new page, click **Create**.

About History

History displays the Recent Changes report. Recent Changes displays recent edits for the current page by developer, application, page ID, modification date, attribute, and action. History lists a history of edited pages by page ID, page name, developer, and edit date.

About Page Detail

Page Detail displays the detailed report describing all currently defined page components and processes. Select the following options at the top of the page to display or hide information:

- **All.** Enabled by default. Displays detailed information about the current page including defined regions, items, buttons, processes, validations, branches, and computations.
- **Regions, Items, and Buttons.** Displays detailed information of all items and button defined in each region of the page.
- **Processes.** Displays details about defined processes including source code.
- **Validations.** Displays any defined validations performed on the current page.
- **Branches.** Displays information about branching performed on the current page.
- **Computations.** Displays details about computations on the current page defined at the page or application level.

About Related Pages

Related Pages displays the Related Pages and Components report. This report displays all components that point to the current page, including menus, branches, tabs, navigation bar icons, and list entries. To edit a component, click the component name.

About Summary of All Pages

Summary of All Pages displays the Page Component Counts report. This report lists all defined components by page.

About Tree View

Clicking **Tree View** displays the Page Tree report. Use this page to view and edit page attributes using the following hierarchy:

```
. Page
. . . Regions
. . . . . Items
. . . . . Buttons
. . . Process
. . . Computations
. . . Validations
. . . Branches
```

To edit an attribute, simply select it.

Using the Developer Toolbar

Users who log in to Oracle HTML DB having developer privileges have access to the Developer toolbar. The Developer toolbar offers a quick way to edit the currently selected page, create a new page, control, or component, view session state, or turn edit links on an off.

As shown in Figure 7–5, the Developer toolbar displays at the bottom of every page in a running application. It offers a quick way to edit the currently selected page, create a new page, control, or component, view session state, or toggle edit links on or off.

Figure 7–5 Developer Toolbar

The Developer toolbar consists of the following links:

- **Edit Application** links you to the Application Builder home page. (See "Viewing a Page Definition" on page 7-5.)
- **Edit Page**.accesses the Page Definition for the currently running page. (See "Editing Page Attributes" on page 7-52.)

- **New** links to a wizard that enables you to create a new blank page, component (report, chart, or form), page control (region, button, or item), or a shared component (menu, list, or tab).
- **Session** links you to session state information for the current page. (See "Viewing Session State" on page 6-13.)
- **Debug** runs the current page in debug mode. (See "Accessing Debug Mode" on page 9-2.)
- **Show edit links** toggles between Show edit links and Hide edit links. Clicking **Show edit links** displays an edit link (resembling four gray dots) to the right of most page components or controls. By clicking an edit link you can edit the selected component or control.

Creating an Application

You create a new application in Oracle HTML DB using the Create Application Wizard. You delete an application from the Application Builder home page.

Topics in this section include:

- Creating a New Application
- Deleting an Application

> **See Also:** "Viewing Application Attributes" on page 7-28 and "Editing Application Attributes" on page 7-29 for more information on application attributes

Creating a New Application

You can use the Create Application Wizard to create a new application having up to nine pages.

To create an application using the Create Application Wizard:

1. Click the **Build** icon.

   Application Builder appears.
2. Click the **Create Application** icon.
3. Choose the method by which you want to create your application:
   - **From Scratch.** Enables you to define tabs, select a user interface (UI), and many other options.

- **Based on an Existing Application.** Creates a copy of another application, including any authentication settings, but without any the pages. Select this option to create an application using the same user interface templates as the application you are copying.
- **Based on Existing Tables.** Creates a complete application based on existing tables you specify. Includes a menu, breadcrumb menus, report page, form page, and a chart page. Select this option if you have a single table from which you would like to create a report and insert, update and delete rows.

4. Follow the on-screen instructions.

Deleting an Application

You can delete an application from the Application Builder home page, or while editing application attributes. If you delete an application you also delete all defined attributes including templates, processes, buttons, and pages.

To delete an application from Application Builder home page:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. From the Tasks list, select **Delete this Application**.
4. Follow the on-screen instructions.

To delete an application while editing application attributes:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. Select the **Edit Attributes** icon.

   The Edit Application Attributes page appears.
4. Verify the application ID.
5. To delete the current application, click **Delete**.

Creating a New Page Using a Wizard

You can create a new page in your application by clicking **Create** on the Page Definition page, selecting the **New...** link on the Developer toolbar, or by using a page wizard.

Topics in this section include:

- About SVG Charting Support
- Creating a Page While Viewing the Page Definition
- Creating a Page from the Developer Toolbar
- Creating a Page Using a Wizard
- Deleting a Page

About SVG Charting Support

Oracle HTML DB supports two types of graphical charts:

- HTML
- SVG

SVG (Scalable Vector Graphics) is an XML-based language for Web graphics from the World Wide Web Consortium (W3C). SVG charts are defined using an embed tag.

When evaluating whether an SVG chart is appropriate chart type for your application remember that:

- Some Web browsers do not support SVG charts
- Most Web browsers that support SVG charts require users download an SVG plug-in

You define charts in Oracle HTML DB using a wizard in which you define a SQL query using the following syntax:

```
SELECT LINK, LABEL, VALUE
FROM   ...
```

`LINK` is an URL, `LABEL` is the text that displays in the bar, and *`VALUE`* is the numeric column that defines the bar size.

For example:

```
SELECT null, ename, sal
```

```
FROM   scott.emp
WHERE  deptno = :P101_DEPTNO
```

Creating a Page While Viewing the Page Definition

To create a new page while viewing a Page Definition:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. Navigate to a specific page, by either:
   - Entering the page ID in the Edit Page field and click **Go**
   - Clicking the page name
4. Under Page in Page Rendering, click **Create**.
5. Follow the on-screen instructions.

> **See Also:** Editing a Page Definition on page 7-38 for more information on editing page attributes

Creating a Page from the Developer Toolbar

Users who log in to Oracle HTML DB having developer privileges have access to the Developer toolbar. The Developer toolbar displays at the bottom every page and offers a quick way create a new page.

To create a new page from the Developer toolbar:

1. On the Developer toolbar, select **New**.

   The Create New Component Wizard appears.
2. Select **Page** and follow the on-screen instructions.

> **See Also:** "Using the Developer Toolbar" on page 7-10

Creating a Page Using a Wizard

Oracle HTML DB includes the following types of wizards for creating pages:

- Wizards that add discrete controls such as regions, items, and validations to an existing page

- Wizards that add an entire component (such as a report or chart) to an existing page
- Wizards that create one or more new pages with components on them

To create a new page using a page wizard:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. Select the **Wizards** tab.

   The Create Page(s) Wizards page appears.
4. Select the type of page or pages you would like to create.
5. Follow the on-screen instructions.

> **See Also:** "Editing a Page Definition" on page 7-38 for more information on editing page attributes

Deleting a Page

You can delete a page while editing page attributes.

To delete a page:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. Navigate to a specific page, by either:
   - Entering the page ID in the Edit Page field and click **Go**
   - Clicking the page name
4. Under Page, click **Edit**.
5. Under Primary Page Attributes, verify the page and application ID.
6. To delete the page, click **Delete** at the top of the page.
7. Follow the on-screen instructions.

See Also: "Editing a Page Definition" on page 7-38 for more information on editing page attributes

Working with Templates

Templates control the look and feel of the pages in your application. Oracle HTML DB includes a number of templates each of which has a distinctive theme.

Topics in this section include:

- Viewing Existing Templates
- Creating Custom Templates
- Editing Templates

See Also:

- "How Application Builder Uses Templates" on page 6-5
- "About User Interface Templates" on page 7-34
- "About Template Defaults" on page 7-35

Viewing Existing Templates

You view and edit templates on the Templates page.

To view a template from the Templates tab:

1. Click the **Build** icon.
2. Select the **Templates** tab.

   The Templates page appears.
3. To view an existing template, click the template name.

You can also access the Templates page from the Shared Components section of Page Definition.

See Also: "Viewing a Page Definition" on page 7-5

About Cascading Style Sheets

A cascading style sheet (CSS) provides a way to control the style of a Web page without changing its structure. When used properly, a CSS separates visual

attributes such as color, margins, and fonts from the structure of the HTML document.

Oracle HTML DB includes page templates with built-in UI themes which reference their own CSS. The style rules defined in each CSS for a particular theme also determine the way reports and regions look. When you copy templates from the Oracle HTML DB gallery, make sure you select templates that match the UI theme you are currently using.

Creating Custom Templates

If you need to create a custom template, it is generally simplest to start with an existing template and then modify it. Once you have created one or more default templates, you can modify those templates to fit your specific needs. The Templates page is grouped by UI themes.

To create a custom template:

1. Click the **Build** icon.
2. Select the **Templates** tab.
3. At the top of the Templates page, click **Create**.
4. Select the type of template you would like to create.
5. Select a creation method:
   - **Start with a default template**
   - **From scratch**
   - **As a copy of an existing template**
6. Follow the on-screen instructions.

You can also make a copy of an existing template by scrolling down to the appropriate template type and click **Create/copy**.

Editing Templates

Once you create a custom template, you can quickly edit it by clicking the edit icon.

To edit an existing template:

1. Click the **Build** icon.
2. Select the **Templates** tab.

3. Locate the template you wish to edit and click the edit icon.
4. Follow the on-screen instructions.

As you edit templates, you can make changes in one window and run your application in another by selecting **Return to page**. Selecting this check box, keeps the page you are editing current after you click **Apply Changes**.

Topics in this section include:

- Editing Page Templates
- Editing Region Templates
- Editing Report Templates
- Editing List Templates
- Editing Label Templates
- Editing Menu Templates
- Editing Button Templates
- Editing Popup LOV Templates

Editing Page Templates

Templates define the appearance of a page. Each template consists of a header template, a body template, a footer template, and a number of subtemplate definitions. If you do not specify a page template as a page level attribute, then the HTML DB engine uses the default page template defined on the Edit Application Attributes page.

Page templates combine static HTML with substitution strings which are replaced at runtime. You use substitution strings to indicate the existence and placement of a component within a page template. You can further specify how a component should display using subtemplates.

Table 7–2 describes the available page template substitution strings. Note that all template substitution strings must be in upper case and begin and end with a pound sign (#).

Table 7–2 Page Template Substitution Strings

Substitution String	Description
#ONLOAD#	May be used in the Header and Footer section of the page template and should be placed inside the <body> html tag. For example: `<body #ONLOAD#>` Use this string as substitute in a JavaScript call to be executed when a page is loaded by the Web browser. The JavaScript to be called can vary for each page.
#TITLE#	Defines the page title. Typically included within HTML title tags.
#NAVIGATION_BAR#	Defines the existence of navigation bar icons. A navigation bar will appear on every page in your application that uses a template which includes this substitution string.You can expand this substitution string using the Navigation bar subtemplate.
#FORM_OPEN#	Specifies where the HTML open form tag `<form>` is placed. You must include this substitution string in order to submit a form. You do not need to code your own form open, the HTML DB engine does it for you.
#NOTIFICATION_MESSAGE#	Enables developers to communicate messages to the user. Defines where a summary of inline error messages is displayed. Inline error message that can be displayed next to a field, or inline in the notification area or both.
#SUCCESS_MESSAGE#	Defines where in the page success and error message appear. If the page process runs without raising an error, then this text displays. You can customize the display of the success message for each template by adding HTML to be displayed before and after the success message.
#BOX_BODY#	Identifies where the Body displays. If the Body is null, then #BOX_BODY# will be used instead.
#TAB_CELLS#	Identifies the display of standard tabs. **See Also:** "About Standard Tab Attributes" on page 7-22 for more information on defining Standard Tab Attributes

Table 7–2 Page Template Substitution Strings

Substitution String	Description
#PARENT_TAB_CELLS#	Identifies the display of parent tabs. Parent tabs require standard tabs. If your application only has one level tabs, you do not need this substitution string. **See Also:** "About Standard Tab Attributes" on page 7-22 for more information on defining Parent Tab Attributes
#NAVIGATION_BAR#	Identifies the display of navigation bar icons. Does not require an HTML table or row to be opened. The navigation bar is intended to display icons for use in meta navigation such as a help or log out icon. You can expand #NAVIGATION_BAR# using the Navigation Bar subtemplate. **See Also:** "About Subtemplate Definitions" on page 7-21 for more information on Navigation Bar subtemplate
#FORM_CLOSE#	If a #FORM_OPEN# is included, then you must include a #FORM_CLOSE# in the header, body, or footer template. #FORM_OPEN# must appear before the #BOX_BODY# and #FORM_CLOSE# must appear after the #BOX_BODY# substitution string.
#REGION_POSITION_NN#	Identifies the exact placement of regions within a page. If no region is specified (for example, #REGION_POSITION_01#) then #REGION_POSITION_01# will be replaced with nothing.
#GLOBAL_NOTIFICATION#	Displays the Global Notification attribute. Global notifications are intended to communicate system status, such as a pending system downtime. You can also use HTMLDB_APPLICATION.G_GLOBAL_NOTIFICATION to set this value if you wish to set it programmatically **See Also:** "About Global Notifications" on page 7-36 for more information on the Global Notification attribute
#HEAD#	Used after the <head> open tag, but before the </head> close tag. You can optionally define the contents of #HEAD# for each page (for example, to reference additional style sheets or JavaScript libraries).

See Also:

- "Using Substitution Strings" on page 6-22
- "Creating a New Page Using a Wizard" on page 7-13

The sections that follow describe specific sections of the Edit Page Template page.

About Template Identification Application ID identifies the application to which this template corresponds. Name identifies the template name used by application developers to identify the template.

About Template Subscription Enables you to apply an existing template to the current application. When you select an existing template, you become a subscriber to that template.

To load a new copy of a master template, click **Refresh**.

About Header, Body, and Footer Definitions Each template consists of a header, a body, a footer, and subtemplate definitions. Use substitution strings to indicate the existence of a component on a page template. All template substitution strings must be in upper case and begin and end with a pound sign (#). You can include the substitution strings listed.

Header is the first section of the page template. When displaying regions or executing processes or computations AFTER HEADER, they will display or execute immediately after this section in the template is rendered.

Body is the second section in the page template and is rendered after the header and before the footer section. At a minimum, you must include the `#BOX_BODY#` substitution string.

Footer is the third section.

About Subtemplate Definitions Use Subtemplate Definitions to specify how a component should display. Available subtemplate definitions include:

- **Success Message.** Expands the `#SUCCESS_MESSAGE#` substitution string. You can define a success message either programmatically, or as an attribute of a process. If a success message exists and if the page template includes the `#SUCCESS_MESSAGE#` substitution string, then this subtemplate is used to render the message.
- **Navigation Bar.** Controls the display of navigation bar icons. This subtemplate is only relevant if the `#NAVIGATION_BAR#` substitution string is identified in your page template. Use the `#BAR_BODY#` substitution string to identify where each navigation bar icon should display.
- **Notification.** Alerts the user that validation errors have occurred. This subtemplate expands the `#NOTIFICATION _MESSAGE#` substitution string.

Notification messages will not be display unless the #NOTIFICATION_MESSAGE# substitution string is included in the page template.

About Standard Tab Attributes You MUST populate this attribute if your application includes standard tabs. Standard tabs may be placed in the header, body, or footer sections of the page template using the #TAB_CELLS# substitution string. The page template Header/Body/Footer defines the HTML table and rows. This subtemplate defines how these tabs display by defining the specific cell. Available attributes include:

- **Current Tab.** Defines the subtemplate to use for standard tabs that are selected (or current) Whether or not a tab is current is determined by standard tab attributes and the page displays. Use #TAB_TEXT# to position a tab's label and link within the template.
- **Non Current Standard Tab.** Defines the subtemplate to use for standard tabs that are not current. Use #TAB_TEXT# to position a tab's label and link within the template.

About Parent Tab Attributes You MUST populate this attribute if your application includes two levels of tabs. Parent tabs may be placed in the header, body, or footer section of the page template using the #PARENT_TAB_CELLS# substitution string. Parent tabs only display in conjunction with standard tabs. Available attributes include:

- **Current Parent Tab.** Defines the subtemplate to use for parent tabs that are selected (or current). Whether or not a tab is current is determined by the page that displays and the standard tab set the page uses. Use #TAB_TEXT# to position a tab's label and link within the template.
- **Non Current Parent Tab.** Defines the subtemplate to use for parent tabs that are not current. Use #TAB_TEXT# to position a tab's label and link within the template.

About Image Based Tab Attributes Use this subtemplate for tabs that are entirely based on images.

About Multi Column Region Table Attribute Each region is assigned a column and a display sequence within that column. When regions are placed in more than one column, Oracle HTML DB renders the regions using an HTML table. This attribute will be used in the <table> tag used for the layout.

About Error Page Template Control Use this attribute only when a page template will be designated as an error template. Use `#MESSAGE#` to place the error message and `#BACK_LINK#` to display a link back to the previous page. A template can be designated as an error template by editing the application attributes. For example:

```
#MESSAGE#

<br>

<a href="#BACK_LINK#">back</a>
```

About Configuration Management You can use build options to enable or disable functionality. Most application attributes have a build option attribute.

Build Options have two possible values: INCLUDE and EXCLUDE. A component that is excluded is not considered part of the application definition at runtime.

> **See Also:** "Using Build Options to Control Configuration" on page 6-11

About Comments Use this attribute to record developer comments.

Editing Region Templates

Regions templates control the appearance and placement of region attributes. Region templates frequently use HTML tables to arrange content.

Region templates apply style elements to regions. Region templates display substitution strings. `#BODY#` (the only required substitution string) identifies where the source of the region should be placed. All other substitution strings are optional. You can use these substitution strings to indicate the existence and placement of a component (such as a button) within the region.

The topics that follow describe specific sections of the Edit Region Template page.

About Region Template Identification **Application ID** Identifies the application to which this template corresponds. **Name** identifies the template name used by developers to identify the template.

About Template Subscription Applies an existing template to the current application. When you select an existing template, you become a subscriber to that template.

To load a new copy of a master template, click **Refresh**.

About Region Template `#BODY#` (the only required substitution string) identifies where the source of the region should be placed. All other substitution strings are optional.

When you create a button in a region position, the positions you have defined will appear in a select list. Use the following substitution strings to define positions for placement of buttons in a region:

- `#EDIT#`
- `#CLOSE#`
- `#CREATE#`
- `#EXPAND#`
- `#HELP#`
- `#DELETE#`
- `#COPY#`
- `#NEXT#`
- `#PREVIOUS#`

See Also: "Using Substitution Strings" on page 6-22

About Form Table Attributes Page items display within regions. Items are rendered as HTML form elements in an HTML table. With this template property, you can define attributes that will be placed in the `<table>` tag. For example:

```
class="tanBox"
```

About Comments Use this attribute to record developer comments.

Editing Report Templates

Report templates are used to apply formatting and style elements to reports. Each report template identifies column names using the syntax `#1#, #2#, #3#` and so on. You can also name columns using column name substitution syntax such as `#ENAME#` or `#EMPNO#`. You can reference any item from your application within your template. For example, to reference an item called *`ABC.`* in your template, you could include the exact substitution string *`&ABC.`*. The actual value of ABC. would be provided by an end user editing an item in your application named *`ABC`*.

Oracle HTML DB includes two types of report templates:

- Generic Column Templates
- Named Column Templates

The sections that follow describe generic column templates, named column templates, as well as how to use report templates conditionally, incorporate Oracle tags, and add JavaScript to row templates.

> **See Also:** Online help for more information on using specific sections of the Edit Report Template page

About Generic Column Templates You can use generic column templates for most queries. This template defines the look of each column. You can conditionally display one-column templates for rows that meet a specific condition and use another column template for other rows. You cannot use this type of template to display multiple columns on the same row differently. If you want each column to use a specific style, you could define the column template similar to the following:

```
<td class="tabledata" align="#ALIGN#">#COLUMN_VALUE#</td>
```

This example assumes your page template includes a CSS containing the class `tabledata`. This example also demonstrates the use the substitution strings `#ALIGN#` and `#COLUMN_VALUE#`. If you actually ran this report, these substitution strings would be replaced with values generated by the results of a SQL query.

If your query uses an expression in the select list, it is a good idea to create an alias for the columns to avoid runtime errors. For example, suppose your query was as follows:

```
SELECT ename, (sal + comm) * 12 FROM emp
```

You could rewrite the query to alias the columns as follows:

```
SELECT ename, (sal + comm) * 12 yearly_comp FROM emp
```

About Named Column Templates Although named column templates offer a great deal of flexibility, you may need to create a new template for each query. When you use a named column template, you specify the column name in the template. For example:

```
<tr><td>#ENAME#</td><td><font color=red>#SAL#</td></tr>
```

You can also include a position notation. The following example demonstrates how to use following HTML and substitution strings:

```
<tr><td>#ENAME#</td><td><font color=red>#SAL#</td></tr>

<tr><td>#1#</td><td><font color=red>#2#</td></tr>
```

About Conditional Use of Report Templates By creating conditions, you can create a report that displays rows differently depending upon who runs the report. You can conditionally use up to four report templates for each query and can refer to specific rows using the syntax *#1#*, *#2#*, *#3#*. For example, to display a row in bold if the salary was greater then 2000 you could include the following row template condition:

```
#5# > 2000
```

About Using JavaScript in Row Templates You can conditionally display HTML depending upon values in the database using JavaScript. The following example displays an HTML row only if the GROUP_DESC query column is not null.

```
<script language="javascript">
IF ( "#GROUP_DESC#" != "" )
document.writeln( "<TR>;
<TD BGCOLOR=#336699>;<FONT SIZE=2 COLOR=#FFFFFF FACE=Arial, Helvetica>Group
Description</FONT></TD>
</TR>
</TR>
<TD>#GROUP_DESC#</TD>
          </TR>" );
</TR>" );
```

See Also:

- Online help for more information on using specific sections of the Edit Report Template page
- "Creating Regions" on page 8-14

Editing List Templates

A list is a shared collection of links. You control the appearance of a list through list templates. Using template attributes, you can also define a list element to be either current or non current for a specific page.

See Also:

- Online help for more information on using specific sections of the Edit List Template page
- "Creating Lists" on page 8-11

Editing Label Templates

Label templates are designed to centrally manage HTML markup of page item labels. Each item can have an optional label. You can control how these labels display using label templates. For example, you could create a label template called "Required Field" which references an image (such as an asterisk) to indicate to the user that the field is required.

Label templates enable you to define a before and after text string that gets prepended and appended to the item.

See Also: Online help for more information on using specific sections of the Edit Label Template page

Editing Menu Templates

A menu template controls the display of menus. You select a menu template when you create a region.

Breadcrumb Style Menu Navigation Breadcrumb style menus usually indicate where the current page is relative to other pages in the application. In addition, users can click a specific page to instantly view it. Oracle HTML DB includes breadcrumb paths beneath the standard tabs (or second level navigation tabs) at the top of each page as shown in the following example:

Figure 7–6 Breadcrumb Style Menu

See Also:

- Online help for more information on using specific sections of the Edit Menu Template page
- "Creating a Menu Template" on page 8-8
- "Creating Menus" on page 8-7

Editing Button Templates

Button templates enable application developers to customize the look and feel of a button. To build a button, you can use multiple images or HTML tags. Using button templates is optional.

Editing Popup LOV Templates

Popup LOV template controls how popup lists display for all items defined as POPUP. You can only specify one popup LOV template for each application.

See Also:

- Online help for more information on using specific sections of the Edit LOV Template page
- "Creating Lists of Values" on page 8-20

Viewing Application Attributes

Application attributes are not specific to one page, but apply to all pages in an application. Once you create an application the next step is to specify application attributes.

To view application attributes:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. Click the **Edit Attributes** icon. (See Figure 7–7.)

Figure 7–7 Edit Attributes Icon

The Edit Application Attributes page appears.

See Also:

- "Editing Application Attributes" on page 7-29
- "Editing a Page Definition" on page 7-38

Editing Application Attributes

Application attributes apply to all pages in an application. Once you create an application the next step is to specify application attributes.

To edit application attributes:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. Select the **Edit Attributes** icon.

   The Edit Application Attributes page appears.

   Oracle HTML DB creates a unique application ID when you create a new application. The application ID displays at the top of the page. Beneath the application ID are links to various sections of the page. Required values are marked with a red asterisk (*).

   The topics that follow describe the specific sections of the Edit Application Attributes page.

Topics in this section include:

- About Application Definition
- About Authorization
- About Session Management

- About User Interface Templates
- About Template Defaults
- About Globalization
- About Application Availability
- About Global Notifications
- About Virtual Private Database (VPD)
- About Static Substitution Strings
- About Build Options
- About Application Comments

About Application Definition

Use this section to define basic characteristics of your application, including the application name, an optional alphanumeric alias, a version number, and the application owner. Table 7–3 describes all Application Definition attributes.

Table 7–3 Application Definition Attributes

Attribute	Description
Name	Provides a short descriptive name for the application to distinguish it from other applications in select lists and reports.
Application Alias	Assigns an alternate alphanumeric application identifier. You can use this identifier in place of the application ID. For example, suppose you create an alias of `myapp` for application 105. Using `f?p` syntax, you could call application 105 as either: ■ `f?p=105:1` ■ `f?p=myapp:1`

Table 7–3 Application Definition Attributes

Attribute	Description
Version	Enables application versioning. You can also automatically tie the version to the date of last modification using the following format masks: ■ YYYY.MM.DD ■ MM.DD.YYYY ■ DD.MM.YYYY If your application version uses YYYY.MM.DD then Oracle HTML DB replaces this format mask with the date of last modification of any application attribute.
Image Prefix	Enter virtual directory set up during installation that points to the actual path to the file system that contains images for your application. Ask your administrator if you are unsure. By default, it is configured as '/i/', your administrator may have customized this option after installation. Do not include a full image path only the virtual directory name. When embedding an image in static text (for example, in page or region headers or footers) you can reference an image using the substitution string #IMAGE_PREFIX#. For example, to reference the image go.gif you would use the following syntax: <img src="#IMAGE_PREFIX#*go.gif*">
Proxy Server	Use this field when you specify a proxy server. You can reference a proxy server using the PL/SQL package variable HTMLDB_APPLICATION.G_PROXY_SERVER. For example: www-proxy.us.oracle.com Application Builder may require a proxy server when using a region source type of URL. The URL region source embeds the results of the URL (that is, the page returned by navigating to the URL) as the region source. If you use a firewall and the target of a URL is on the opposite side of the firewall from where Oracle HTML DB is installed, you may need this proxy setting.
Default Parsing Schema	Specifies the schema that all SQL and PL/SQL in the application will be parsed as. You may use #OWNER# to reference this value in SQL queries and PL/SQL (for example, in a region or a process).

Table 7–3 Application Definition Attributes

Attribute	Description
Logging	Determines whether user activity is recorded in the Oracle HTML DB activity log. When set to **Yes**, every page view will be logged, allowing a Workspace administrator to monitor user activity for each application.

Exact Substitutions	Select whether or not to use exact substitutions. For optimal runtime performance, it is recommended you use exact substitutions.
	Exact substitutions use the following sytnax:
	`&ITEM.`
	Non-exact substitutions use the following sytnax.
	`&ITEM`

See Also:

- "Using Substitution Strings" on page 6-22
- "Using f?p Syntax to Link Pages" on page 6-20 for more information on linking pages

About Authorization

Use the Authorization field to specify an authorization scheme for your application. You may assign only one authorization to a given application.

When you use an authorization scheme, users can only view those pages that pass the specified authorization scheme. An authorization scheme is a binary operation that either succeeds (equals true) or fails (equals false). If it succeeds then the component can be viewed, if it fails then the component cannot be viewed or processed. If you have attached an authorization scheme to a page and it fails then an error message displays. When you attach an authorization scheme to any other component (such regions, button, or items) and it fails, no error page displays. Instead, the component either does not display or is not processed or executed.

See Also: "Creating an Authentication Scheme" on page 10-10 authorization schemes

About Session Management

Use these attributes when establishing your authentication and session management infrastructure. Table 7–4 describes all session management attributes.

Table 7–4 Session Management Attributes

Attribute	Description
Home Link	This is the relative URL used to display the home page of your application. For example, `f?p=6000:600` could be for application 6000 with a home page ID of 600. The value you enter here replaces `#HOME_LINK#` substitution string in application templates. You can also use this attribute to name a procedure. For example, you could create a procedure such as `personal_calendar` which calls an HTML page to serve as the application home.
Login URL	Specifies the location of the application login page. **See Also:** "Using Substitution Strings" on page 6-22 and "Creating an Authorization Scheme" on page 10-17 for more information
Public User	Identifies the Oracle schema (that is, the user) used to connect to the database when generating unprotected pages. When a user logs in as this user then the HTML DB engine considers this to be a "public user" session. The HTML DB engine supports the following built -in display conditions: ■ `USER_IS_PUBLIC_USER` ■ `USER_IS_NOT_PUBLIC_USER` If the current application user (or `v('USER')`) equals the value of this attribute, then the user is logged on as a public user. Some applications have public (not logged in) and a private (logged in) modes. By determining if the user is a public user, you can conditionally display or hide information. **See Also:** "Establishing User Identity Through Authentication" on page 10-9 for more information For example you can show a login button if the user is a public user and a logout link if the user is not the public user. The public user (if null) defaults to "`PUBLIC_USER`". Reference this value using `HTMLDB_APPLICATION.G_PUBLIC_USER`. The HTML DB engine also has built in condition types `'USER_IS_PUBLIC_USER'` and `'USER_IS_NOT_PUBLIC'` user.

To view details about a selected authentication scheme, click **mange** next to **Authentication: SCHEME**.

About User Interface Templates

Use these attributes to control the look and feel of the pages in your application. An application can have any number of page templates. You can specify a unique template for each page, or if you do not specify a template the HTML DB engine uses the template specified here. Table 7–5 describes the available user interface templates attributes.

Table 7–5 User Interface Templates Attributes

Attribute	Description
Default Page Template	Specifies the default page template for displaying pages. If a page developer does not explicitly choose a template then the HTML DB engine uses the template specified here.
Print Mode Page Template	Identifies the template to be used when the HTML DB engine is in printer friendly mode. When calling the HTML DB engine to render a page, you have the option to identify a printer friendly attribute with values of YES and NO. If you specify YES, then the page displays using a printer friendly template. The HTML DB engine displays all text within HTML Form Fields as text. The printer friendly template does not need to have the `#FORM_OPEN#` or `#FORM_CLOSE#` tag. The objective is to be able to display information with few tables and in a format suitable for printing.
Error Page Template	Optional. Specifies a page template to use for errors that display on a separate page as opposed to those that display inline.
Body Width	When Application Builder creates a table to display the body of the application, this attribute specifies the width it should it use. The default is 98 percent (98%). Typically the developer controls all table creations using region templates. In the event that such templates are unavailable, Oracle HTML DB uses this value.

See Also: "Working with Templates" on page 7-16 and "Editing Templates" on page 7-17

About Template Defaults

Use the following attributes to specify the default template when you create new regions, labels, reports, lists, menus, and buttons. You can override these settings on the edit page for each component.

See Also:

- Online help for more information on each default template setting
- "Working with Templates" on page 7-16 and "Editing Templates" on page 7-17

About Globalization

The attributes described in Table 7–6 enable you to specify globalization options such as the primary application language.

Table 7–6 Globalization Attributes

Attribute	Description
Application Primary Language	Identifies the language in which an application is developed. This language is the base language from which all translations are made. For example, suppose application 100 was authored in English and then translated it into French and published as application 101. The application ID would be transparent to the end user. All modifications to the application should be made to the primary language specified here.
Application Language Derived From	When running an application, the HTML DB engine needs to know the user's language preference so it can set the database language and run the translated application. The database language setting determines date display and sorting characteristics. This option enables you to disable browser derived language support. You also have the option of having the application language derived from an application preference.

See Also: "About Translating an Application and Globalization Support" on page 16-1 for more information on translating an application and globalization support

About Application Availability

Use these attributes to manage your application by defining an application status and build status. For example, if you select the status **Restricted Access**, you can specify which users who have access and can run this application.

See Also: Online help for more information on each Application Availability attribute

About Global Notifications

You can use a global notification to communicate system status. If your page template contains a `#GLOBAL_NOTIFICATION#` substitution string, then the text entered here displays on each page.

See Also: "Using Substitution Strings" on page 6-22

About Virtual Private Database (VPD)

VPD provides an application programmatic interface (API) which enables developers to assign security policies to database tables and views. Using PL/SQL, developers can create security policies with stored procedures and bind the procedures to a table or view by means of a call to an RDBMS package. Such policies are based on the content of application data stored within the database, or based on context variables provided by Oracle database. In this way, VPD permits access security mechanisms to be removed from applications, and centralized.

The PL/SQL you enter in this field is executed immediately after the user is authenticated. v('USER') is accessible from this function. Session state for the current call is not yet initialized when this call is made. If your application does not need to employ VPD to support multiple customers in the same database then leave this attribute null.

See Also:

- "Providing Security Through Authorization" on page 10-17
- *Oracle Label Security Administrator's Guide*

About Static Substitution Strings

Static substitution string are strings that are defined centrally. They are typically used for phrases or labels that occur in many places within an application. Defining

Static substitution strings centrally enables you to affect multiple changes to labels by making a single change to a substitution string.

> **See Also:** "Using Substitution Strings" on page 6-22 for more information on the built-in substitution strings available in Oracle HTML DB

About Build Options

Use build options to enable or disable functionality. Most application attributes have a build option attribute. Do not specify a build option unless you plan on excluding that object from specific installations. Build Options have two possible values: `INCLUDE` and `EXCLUDE`. An attribute that is excluded is treated as if it does not exist.

> **See Also:** "Using Build Options to Control Configuration" on page 6-11

About Application Comments

Use this attribute to record developer comments about the current application.

Viewing Page Attributes

Page attributes only apply to a specific page. Once you have defined application attributes the next step is to specify page attributes. You access page attributes from the Page Definition page.

To view page attributes:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. Navigate to a specific page, by either:
   - Entering the page ID in the Edit Page field and click **Go**
   - Clicking the page name

   The Page Definition appears.

   By default, each page is divided into three sections.

- **Page Rendering.** Includes all attributes for the page, regions, buttons, items, as well as page rendering computations and processing.
- **Page Processing.** Enables you to specify page level application logic such as computations, validations, processes, and branching.
- **Shared Components.** Displays common components that display on every page within an application. Parent tabs, standard tabs, navigation bars, list of values, menus, lists, and templates are examples of shared components. You can hide shared components by disabling the Shared Components customization option.

Attributes found under the section **Page** are the actual attributes of the page. Page Rendering contains attributes that are subordinate to the page (such Regions, Buttons, Items, Computations and Processes). You define these attributes on their own respective pages.

See Also:

- "Editing a Page Definition" on page 7-38 for more information on editing page attributes
- "Building Application Components" on page 8-1

Editing a Page Definition

A page is the basic building block of an application. You create a new application using the Create Application Wizard. Once created, each page has page ID, a name, and typically some text attributes such as a header, title and footer. You add content to your page by creating regions, items and buttons. Page templates and page region templates control the exact look and feel of each page.

You can view the details about a specific component by selecting the component name. You can view, edit, and create new page components by clicking the following buttons:

- **View.** Click this button to view and edit details about currently defined components.
- **Copy.** Click this button to make a copy of current component.
- **Create.** Click this button to create a new component.

Topics in this section include:

- Managing Page Rendering Components

- About Page Processing Components
- Editing Page Attributes

See Also:

- "Understanding the Definition of a Page" on page 7-2
- "Creating an Application" on page 7-11
- "Creating a New Page Using a Wizard" on page 7-13

Managing Page Rendering Components

Under Page Rendering, you specify all page attributes as well as attributes for defined regions, buttons, items, page rendering computation, and page processes.

Topics in this section include:

- About Page
- About Regions
- About Buttons
- About Items
- About Page Computations
- About Page Processes

About Page

Use Page to edit high-level page attributes such as the page name, an optional name alias, and view information about defined tab sets, specified templates, and authorization schemes.

See Also: Editing Page Attributes on page 7-52

About Regions

A region is a section of a page that contains content. The content of a region is determined by the region source. For example, a region may contain a report based on a SQL query you define, or it may contain static HTML.

For more information on specific region attributes, see the on-screen instructions.

Understanding Region Source Types Each region has a region source type. The HTML DB engine interprets a region differently based the type you select. Table 7–7 describes available region types.

Table 7–7 Region Types

Region Type	Description
HTML	HTML regions contain HTML you provide. They are also used as containers for items. Any HTML you type may contain substitution strings.
Report	Report regions can be defined by a SQL query you write, or by using a wizard to guide you through the steps needed to write a query. **See Also:** "Creating Reports" on page 8-28
Chart	Chart regions contain line, bar, or pie charts based on SQL queries. **See Also:** "Creating Charts" on page 8-35
List	List regions are used for navigation and may consist of links or images. Individual list entries can be conditionally displayed. **See Also:** "Creating Lists" on page 8-11
Menu	Menu regions are navigational controls consisting of predefined, hierarchically organized links. **See Also:** "Creating Menus" on page 8-7
PL/SQL Dynamic Content	Regions based on PL/SQL enable you to render any HTML or text using the PL/SQL Web Toolkit.
Other, HTML Text (escape special characters).	Same as HTML region, but the HTML DB engine escapes special characters before they are rendered.
Other, HTML Text (with shortcuts)	Same as HTML region, but with support for shortcuts. **See Also:** "Utilizing Shortcuts" on page 8-40
Other, Help Text	Help regions enable you to provide page level help. **See Also:** "Creating a Help Page" on page 8-41
Other, URL	URL based regions obtain their content by calling a Web server using a predefined URL. **See Also:** "About Regions Based on an URL" on page 8-17

Table 7–7 Region Types

Region Type	Description
Other, Web Service Result	Web service based regions call a predefined Web service to obtain an XML document. An optional XSLT style sheet may be applied. **See Also:** "Implementing Web Services" on page 12-15
Other, Tree	Trees are a hierarchical navigational control based on a SQL query executed at runtime. It enables the user to expand and collapse nodes. **See Also:** "Creating Trees" on page 8-10

See Also:

- "Creating Regions" on page 8-14 for more information on creating specific types of regions
- *Oracle Database Application Developer's Guide - Fundamentals* for more information on developing Web applications with PL/SQL
- *PL/SQL Packages and Types Reference* for more information on htp packages

Controlling Region Positioning When you create a region, you must specify its position or Display Point on the page. Regions are rendered in order of sequence number within a Display Point. You can choose either a default position (such as Page Template Body) or a user defined position in the template (such as Page Template Region Position 1.) In addition to Display Point, you can specify which column the region will be placed in. When you place regions in multiple columns, Oracle HTML DB automatically renders the necessary HTML to produce a multi column layout.

Displaying a Region Conditionally Whether a region is displayed can be based on conditions defined at the region level. Conditions may reference session state, the currently logged in user, or HTML DB engine settings (such as whether or not a page is in Print View mode).

Specifying a Region Header and Footer In addition to the body content of a region (which can be a report, a chart, or HTML with form elements), you can specify additional HTML to be placed above and below a region or in its header and footer. The region footer supports the following substitution strings:

- #TIMING# shows the elapsed time in seconds used when rendering a region. You may use this for debugging purposes.
- #ROWS_FETCHED# shows the number of rows fetched by the Oracle HTML DB reporting engine. You may use these substitution strings to display customized messages to the user. For example:

```
Fetched #ROWS_FETCHED# rows in #TIMING# seconds.
```

- #TOTAL_ROWS# displays the total number of rows that satisfy a SQL query used for a report.
- #FIRST_ROW_NUMBER# and #LAST_ROW_NUMBER# displays the range of rows displayed. For example:

```
Row(s) #FIRST_ROW_NUMBER# through #LAST_ROW_NUMBER# of #ROWS_FETCHED#
displayed
```

About Buttons

As you design your application you can use buttons to direct users to a specific page or URL, or to post or process information. Buttons can be placed in predefined region template positions or among items in a form.

For more information on specific button attributes, see the on-screen instructions.

About Branching with Buttons Each page can include any number of branches. A branch links to another page in your application or to an URL. The HTML DB engine considers branching at different times during page processing. You can choose to branch before processing, before computation, before validation, and after processing. Like any other component in Application Builder, branching can be conditional. For example, you can branch when a user clicks a button. When you create a branch, you associate it with a specific button. The branch will only be considered if a user clicks the button.

See Also:

- "About Branching" on page 7-42
- "Creating a Branch" on page 8-13
- "Using the Create Button Wizard" on page 8-19

Understanding the Relationship Between Button Names and REQUEST The name you give a button determines the value of the built-in attribute REQUEST. You can reference

the value of REQUEST from within PL/SQL using the bind variable `:REQUEST`. By using this bind variable, you can conditionally process, validate, or branch based on which button the user clicks. You can also create processes that execute when the user clicks a button. You can also use a more complex condition as demonstrated in the following examples:

```
If :REQUEST in ('EDIT','DELETE') then ...
If :REQUEST != 'DELETE' then ...
```

These examples assume the existence of buttons named EDIT and DELETE. You can also use this syntax in PL/SQL Expression conditions. Be aware, however, that the button name case is preserved. In other words, if you name a button "LOGIN" then a request looking for the name "Login" will fail. For example:

```
<input type="BUTTON" value="Finish" onClick="javascript:doSubmit('Finish');">
```

Note that *`Finish`* is the name of the REQUEST and in this example is case sensitive.

Displaying Buttons Conditionally You can choose to have a button display conditionally by making a selection from the Conditional Display Type list and entering an expression.

> **See Also:** "Creating Buttons" on page 8-19 for more information on creating specific types of buttons

About Items

An item is part of an HTML form and can be a text field, text area, password, combobox, checkbox, and so on. There are two types of items, page items and application items. Page items are placed on a page and have associated user interface properties, such as Display As, Label and Label Template. Application items are not associated with a page and therefore have no user interface properties. An application item can be used as a global variable.

For more information on specific item attributes, see the on-screen instructions.

> **See Also:** "Using Substitution Strings" on page 6-22

Understanding Item Display As Options For each item, you specify a type by making a selection from the Display As list. Table 7–8 describes many available Display As options.

Table 7–8 Item Display As Options

Display As Option	Description
Button	Used to build forms in Oracle HTML DB. Use an Item Button when you want to place a button among other fields (or items) in a form. When clicked, this type of button will automatically switch the HTML DB engine to processing mode, enabling you to perform validations, execute processes, or branch the user to another page. **See Also:** "About Buttons" on page 7-42

Check box	Displayed using a list of values. A list of values is required for items displayed as check boxes. The value corresponding to a checked box is returned in a single colon (:) delimited string.
	
	The following example demonstrates how to create a single check box that returns `YES`. This example would display both a check box and a field label.
	
	`SELECT NULL display_text, 'YES' return_value FROM DUAL;`
	
	This example includes the additional text "Click to select."
	
	`SELECT 'Click to select' display_text, 'YES' return_value FROM DUAL;`
	
	See Also: "HTMLDB_UTIL" on page 13-1 for more information on breaking up returned values

Date Picker	Displays a text field with a calendar icon next to it. When clicked, this icon displays a small calendar from which the user can select a date and a time (optional). If the format you need is not included in the Display As list, select **Date Picker (use applications format mask)**. When using a format mask, your application looks for the format in an item called `PICK_DATE_USING_APP_FORMAT_MASK`. Note that you need to populate this item before this item type will work.
Display as Text (based on LOV, does not save state)	Displays a read-only version of a display value from a list of values by using the item's value in session state to look up the corresponding display value in the associated list of values. The value displayed on the screen is not saved in session state upon submit.
Display as Text (based on LOV, saves state)	Displays a read-only version of a display value from a list of values by using the item's value in session state to look up the corresponding display value in the associated list of values.
Display as Text (escape special characters, does not save state)	Displays a read-only version of the value in session state, escaping special characters. Session state is not saved.
Display as Text (saves text)	Displays a read-only version of the value in session state. Upon submit, the value displayed is saved in session state.

Table 7–8 Item Display As Options

Display As Option	Description
File	Displays a text field with a "Browse..." button. This enables the user to locate a file on a local file system and upload it. Oracle HTML DB provides a table for these files to be uploaded to as well as an API to retrieve the files.
Hidden	Renders an HTML hidden form element. Session state can be assigned and referenced just like a text field.
List Managers	Based on a list of values. This item enables you to manage a list of items by selecting and adding to a list. The list of values display as a popup.
Multiselect List	Renders as a multiselect HTML form element. When submitted, selected values are returned in a single colon delimited string. You can break up the values using the `HTMLDB_UTIL API`. **See Also:** "HTMLDB_UTIL" on page 13-1
Password	Renders as an HTML password form element.

Popup LOV	Renders as a text field with an icon next to it from which a user can select a value from a popup window. The list in the popup window is driven by a list of values. There are two types of Popup LOVs, one that fetches a set of rows when the window pops up and one that does not.
	Popup LOV values must select two columns. For example:
	`SELECT ename, empno FROM emp`
	If one of the columns is an expression, remember to use an alias. For example:
	`SELECT ename||' '||job display_value, empno FROM emp`

Radio Group	Renders as an HTML radio group form element, based on a list of values. Choose **Radiogroup with Submit** to have the page submitted when the radio button is selected. The following example displays employee names (`ename`), but returns employee number (`empno`)s: `SELECT ename, empno FROM emp`

Table 7–8 Item Display As Options

Display As Option	Description
Select List	Displays using a list of values. A list of values is required for items displayed as a select list. Select lists are rendered using the HTML form element `<select>`. The values in a select list are determined using a shared list of values or a list of values defined at the item level. You may specify the NULL display value and NULL return value. The following example would return employee names (`ename`) and employee numbers (`empno`) from the `emp` table. Note that column aliases are not required and are included in this example for clarity. `SELECT ename display_text, empno return_value FROM emp` Oracle HTML DB provides additional enhancements to a standard HTML select list: ■ **Select List with Submit** - Submits the page when the user changes its selected value. Upon submit, the REQUEST will be set to the name of the item that represents the select list, allowing you to execute conditional computations, validations, processes, and branches. ■ **Select List with Redirect** - Redirects the user back to the same page, setting ONLY the newly selected value of the select list in session state. ■ **Select List Returning URL redirect** - Based on a list of values with URLs as the return values. Changing the value of the select list causes the browser to redirect to the corresponding URL. ■ **Select List with Branch to Page** - Based on list of values with page IDs as return values. Changing the selected value in the select list causes the HTML DB engine to branch to the corresponding page.
Stop and Start HTML Table (Displays label only)	Oracle HTML DB uses HTML tables to render items. This item may be used to control the layout of items in forms by closing a table and starting a new one.

Table 7–8 Item Display As Options

Display As Option	Description
Text	Displays as an HTML text field containing a maximum of 30,000 bytes of text. You control the maximum length and display width by editing the Height and Width item attribute.
Text Area	Renders as an HTML text area. This is no maximum length for an item displayed as a text area. You control the height and width by editing the Height and Width item attribute. Additional available Text Area Display As options include: ▪ **Text Area (auto height)** - Varies the height based on the amount of text. Use this option to have a large text area when you have a lot of data and a smaller text area when you have little or no data. ▪ **Text Area with Counter** - Includes a counter that displays the number of bytes entered in the field. ▪ **Text Area with Spell Checker** - Provides a popup English language spell checker. ▪ **Text Area with HTML Editor** - Provides basic text formatting controls. Note that these controls may not work in all Web browsers.
Text with Calculator	Renders as a text field with an icon next to. When clicked, the icon displays a small window containing a calculator. Calculations are placed back in the text field.

Referencing Item Values You can reference item values stored in session state in regions, computations, processes, validation, and branches. Table 7–9 describes the supported syntax for referencing item values.

See Also: "Managing Session State Values" on page 6-14

Table 7–9 Syntax for Referencing Item Values

Type	Syntax	Description
SQL	`:MY_ITEM`	Standard bind variable syntax for items no longer than 30 bytes. Use this syntax for references within a SQL query and within PL/SQL.
PL/SQL	`v('MY_ITEM')`	PL/SQL syntax referencing the item value using the v function. **See Also:** "Oracle HTML DB APIs" on page 13-1
PL/SQL	`nv('MY_NUMERIC_ITEM')`	Standard PL/SQL syntax referencing the numeric item value using the nv function. **See Also:** "Oracle HTML DB APIs" on page 13-1
Static Text	`&MY_IITEM`	Static text.
Static Text (exact)	`&MY_IITEM.`	Static text. Exact Substitution.

You can set the value of an item in your application using any of the following methods:

- For page items, use the Source Attribute to set the item value.

  From the Page Definition, select the item name to view the item attributes (called Edit Page Item). Scroll down to the Source attribute and edit the appropriate fields.

  You can also set the value of an item in any region based on PL/SQL or a process using the following syntax:

  ```
  BEGIN
   :MY_ITEM :=  'new value';
  END;
  ```

- Pass the value on an URL reference using `f?p` syntax. For example:

  ```
  f?p=100:101:10636547268728380919::NO::MY_ITEM:ABC
  ```

- Set the value using a computation. Computations are designed to set item values. For example:

  ```
  TO_CHAR(SYSDATE,'Day DD Month, YYYY');
  ```

- Use the PL/SQL API to set an item value within a PL/SQL context. For example:

```
HTMLDB_UTIL.SET_SESSION_STATE('MY_ITEM',SYSDATE);
```

See Also:

- "Clearing Session State" on page 6-16
- "Oracle HTML DB APIs" on page 13-1

Displaying Conditional or Read-only Items You can choose to have an item display conditionally by making a selection from the Conditional Display, Conditional Display Type list and entering an expression.

You can define an item a read-only by making selections under Items Is Read-only When.

About Page Computations

You can use page computations to assign a value to an identified item when a page is submitted or displayed. Application level computations can also be used to assign values to items. Most application level computations are performed for every page in an application. In contrast, computations created at the page level only execute when that page is rendered or processed.

Specifying an Item and Computation Type For each computation, specify the item for which you are creating the computation as well as a computation type.

Defining a Computation Point and Computation Source You control when a computation executes under Computation Points attributes by specifying a sequence number and a computation point. The computation point On New Instance executes the computation when a new session (or instance) is generated.

Under Computation Source, enter an expression or query to compute an item's value. In the event a computation fails, you can optionally define an error message in Computation Error Message field.

Creating Conditional Computations You can make a computation conditional by making a selection from the Conditional Type list and entering text in the expression fields.

About Page Processes

You create a page process to execute some type of code (such as SQL or PL/SQL) or to make a call to the rendering engine. You would typically create a page process to alter data in some way (for example, to perform an UPDATE, INSERT, or DELETE).

A page process is a unit of logic that runs when a specific event occurs, such as loading or submitting page, resetting session state, and automatic row processing. From a functional perspective, there is no difference between page level and application level processes. The difference lies in the point at which the process occurs.

For more information on specific page processing attributes, see the online help.

Creating a Page Process You create a process by running the Create Page Process Wizard. During the page creation process, you define a process name, specify a sequence, the point at which process will execute, and a process type. You can change nearly all of these attributes on the Edit Page Process page.

Defining Processing Points and Source You control when a process executes under Process Firing Point attributes by specifying a sequence number and a process point. You can prevent a process from running during subsequent visits to a page by selecting one of the following options under Run Process:

- Once for each page visit
- Once for each session or when reset

Under Source, enter the appropriate code for SQL or PL/SQL process types. In the event a process fails, you can optionally define an error message in the Process Error Message field.

Creating Conditional Processes

You can make a process conditional by selecting a conditional type and entering an expression under Conditional Processing attributes.

About Page Processing Components

Under **Page Processing**, you specify application logic such as computations, validations, processes, and branching. The HTML DB engine runs this logic in the order it appears on the Page Definition.

For more information on specific page processing attributes, see the online help. Topics in this section include:

About Validations

You can define a validation declaratively by selecting a built-in validation type or by entering custom SQL or PL/SQL. You enter the actual validation edit check in the Validation Messages field. Keep in mind that the validation you enter must be consistent with validation type you selected. For more information on validation types, see online help.

Defining How Validation Error Messages Display You can choose to have validation error messages display inline (that is, on the page where the validation is performed) or on a separate error page.

To define how a validation error message displays:

1. In the **Error Message field**, enter your error message text.
2. From the **Error message display location** list, select a display location.
3. If you selected either **Inline with Field** or **Inline with Field and in Notification** in step 2, select the item associated with the error message from the **Associated Item list**.

Processing Validations Conditionally You can control when and if a validation is performed under Conditional Validation Processing. To have a validation performed when a user clicks a button, make a selection from the When Button Pressed list.

You can add other conditions by making a selection from the Conditional Type list and entering text in the expression fields.

About Branching

A branch is an instruction to link to a specific page, procedure, or URL. For example you can branch from page 1 to page 2 after page 1 is submitted.

About Branch Types You create a new branch by running the Create Page Branch Wizard and specifying Branch Point and Branch Type. The Branch Type defines the type of branch you are creating. For more information on Branch Types, see online help.

Defining a Branch Point and Branch Action You specify when to create a branch by making a selection from the Branch Point list. Valid options include:

- **On Submit: Before Computation** - Occurs before computations, validations, or processing. Use this option for buttons that do not need to invoke any processing, for example, a "Cancel" button.
- **On Submit: Before Validation** - Occurs after computations, but before validations or processing. If a validation fails, page processing stops, a rollback is issued, and the page displays the error. Because of this default behavior, you do not need to create branches to accommodate validations. However, you might want to branch based on the result of a computation (for example, to the previous branch point).
- **On Submit: Before Processing** - Occurs after computations and validations, but before processing. Use this option to branch based on a validated session state, but before performing any page processing.
- **On Submit: After Processing** - Occurs after computations, validations, and processing. This option branches to an URL or page after performing computations, validations, and processing. When using this option, remember to sequence your branches if you have multiple branches for a given branch point.
- **On Load: Before Header** - Occurs before a page is rendered. This option displays another page instead of the current page or redirects the user to another URL or procedure.

Depending upon the Branch Type you select, you can specify the following additional information in Branch Action attributes:

- The page ID of the page you wish to branch to
- PL/SQL code
- An URL address

Branching Conditionally Like other Application Builder components, branches can be made conditional. To create a conditional branch, make a selection from the Conditional Type list and enter text in the expression fields to implement the condition type you choose.

Editing Page Attributes

Page attributes only apply to a specific page. You access page attributes from the Page Definition page.

To edit page attributes:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. Navigate to a specific page, by either:
   - Entering the page ID in the Edit Page field and clicking **Go**
   - Clicking the page name
4. Under Page, click **Edit** to modify existing page attributes.

Required values are marked with a red asterisk (*). For more information on specific page attributes, see the online help.

Topics in this section include:

- About Primary Page Attributes
- About HTML Header
- About Page Header, Footer and Text Attributes
- About On Load JavaScript
- About Security
- About Duplicate Page Submission Checks
- About On Error Text
- About Page Help Text
- About Comments

> **See Also:** "Creating a New Page Using a Wizard" on page 7-13 for more information on creating a new page

About Primary Page Attributes

Use these attributes to define general attributes for the current page such as a page name, an optional alphanumeric alias, and a browser title. Table 7–10 describes editable Primary Page attributes.

Table 7–10 Primary Page Attributes

Attributes	Descriptions
Name	Identifies the name of the current page. This name is used in numerous Oracle HTML DB pages and reports, along with the page ID and page title.
Page Alias	Enter an alphanumeric alias for this page. For example, if you were working on page 1 of application 100, you could create an alias called "home." You could then access this page from other pages using the following `f?p` syntax: `f?p=100:home`
Standard Tab Set	Select a tab set to be used for the first level of tabs for this page.
Title	Enter the title used that displays in the browser window. This title is inserted between the following HTML title tags: `<TITLE></TITLE>`
First Item	Select `AUTO_FIRST_ITEM` to enable JavaScript that sets the focus to the first item on the page when the page displays. Select `NO_FIRST_ITEM` to bypass this behavior
Page Template	Select a page template to control the appearance of this page. This overrides the application template.

About HTML Header

Use this attributes to replace the `#HEAD#` substitution string in the page template header. The values entered here are inserted after the HTML `<HEAD>` tag. You can use attribute to:

- Code page specific inline cascading style classes
- Add additional style sheets for a specific page
- Code page specific JavaScript
- Code page specific meta tag page refresh

About Page Header, Footer and Text Attributes

Use these attributes to define page header, body header, body footer, and page footer text. Table 7–11 describes editable primary page attributes.

Table 7–11 Page Header, Footer and Text Attributes

Attribute	Description
Header Text	Displays after the page template header and before page template body.
Body Header	Displays before showing regions. Displays before the page template `#BOX_BODY#` substitution string.
Body Footer	Displays after showing regions. Displays after page template `#BOX_BODY#` substitution string.
Footer	Displays after page template body and before page template footer.

About On Load JavaScript

Use this attribute to add onload events such as calls to JavaScript. To use this feature, your page template must include `#ONLOAD#`.

You can use the Page HTML Body Attribute to exercise exact control over generated HTML. Use this feature to write into the contents of the `BODY` tag. For example, your page template could define the following:

```
<html.
<head>
...
</head.
<body #ONLOAD# >
```

About Security

Use this attribute to specify an authorization scheme and an authentication method for the current page.

Select the authorization scheme to be applied to the page from the Authorization Scheme list. Authorization schemes are defined at the application level and can be applied to many elements within the application. A given authorization scheme is set up to be evaluated either once for each application session (at session creation) or once for each page view. If the selected authorization scheme evaluates to true, then the page displays (that is, subject to other defined conditions). If it evaluates to false, then the page will not display and an error message appears.

From the Authentication list, specify whether this page has been defined as public or requires authentication. If a page is identified as public, the page can be viewed before authentication. This attribute only applies if the application uses SCHEME

authentication. The application's page sentry function may access this page attribute to identify pages that do not require prior authentication to view. The implementation of the authentication scheme's page sentry function determines whether this PAGE_IS_PUBLIC page attribute has any effect.

About Duplicate Page Submission Checks

Use the Allow duplicate page submissions list to specify whether Oracle HTML DB allows users to process a page multiple times. This can happen when a user clicks the browser's back button and then submits the page again, or if the user clicks the browser's reload button.

Setting this attribute to No, prevents duplicate page submissions.

About Configuration Management

Build options allow you to enable or disable functionality. Most application attributes have a build option attribute. Do not specify a build option unless you plan on excluding that object from specific installations. Build Options have two possible values: `INCLUDE` and `EXCLUDE`. An attribute that is excluded is treated as if it does not exist.

> **See Also:** "Using Build Options to Control Configuration" on page 6-11

About On Error Text

Use this attribute to specify the error text that displays in the `#NOTIFICATION_MESSAGE#` substitution string included in the page template.

> **See Also:** "Editing Page Templates" on page 7-18

About Page Help Text

Use this attribute to enter help text for the current page. Page level help supports shortcuts using the following syntax:

```
"SHORTCUT_NAME"
```

Help text is displayed using a help system that you must develop. To show the help for a specific page, call the `WWV_APPLICATION.HELP` procedure from a page that you create for displaying help text. For example, you could use a navigation bar icon similar to:

```
f?p=4000:4600:&SESSION::&DEBUG::LAST_STEP:&APP_PAGE_ID
```

In the previous example, page 4600 calls the `HTMLDB_APPLICATION.HELP` procedure and passes the page ID for which help will be displayed.

See Also: "Creating a Help Page" on page 8-41

About Comments

Use this attribute to record developer comments about the current page.

Running a Page

The HTML DB engine dynamically renders and process pages based on data stored in database tables. To view a rendered version of your application, you run or submit it to the HTML DB engine. As you create new pages you can run them individually, or run an entire application.

To run page from the Page Definition:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. Navigate to a specific page.

   The Page Definition appears.
4. In the left navigate pane, click **Run Page**.

To run an entire application:

1. Click the **Build** icon.
2. From the Available Applications list, select the application and click **Go**.
3. Click the **Run** icon.

8

Building Application Components

This section provides information about how to build different types application components in Application Builder.

This section contains the following topics:

Displaying Components on Every Page

Page zero of your application functions as a master page. The HTML DB engine renders all components you add to page zero on every page within your application. You can further control whether the HTML DB engine renders a component or runs a computation, validation, or process by defining conditions.

To create a page zero:

1. Create a new page.
2. Specify the page ID as zero (0).

See Also:

- "Creating a New Page Using a Wizard" on page 7-13 for more information on creating a page
- "Understanding Conditional Rendering and Processing" on page 6-9
- "Available Conditions" on page A-1

Adding Navigation

When you build an application you can include a number of different types of navigation including tab sets, navigation bars, menus, trees, and lists. This section describes how to implement navigation in your application.

Topics in this section include:

- Creating Tab Sets
- Creating a Navigation Bar
- Creating Menus
- Creating Trees
- Creating Lists
- Creating a Branch

Creating Tab Sets

Tabs are an effective way to navigate users between pages of an application. You can create a tabbed application look by using parent tabs, standard tabs, and Oracle HTML DB lists.

Application Builder includes two different types of tabs:

- Standard tabs
- Parent tabs

An application having only one level of tabs uses a standard tab set. A standard tab set is associated with a specific page and page ID. You can use standard tabs to link users to a specific page. A parent tab set functions as a container to hold a group of standard tabs. Parent tabs give users another level of navigation as well as a context (or sense of place) within the application. You can use parent tabs link users to a specific URL associated with a specific page.

The topics that follow describe how to add tab sets to your application.

Note: When running the Create Application Wizard, you have the option of creating an application with tabs. The following procedures assume you have already created an application that does not have any tabs.

See Also: "Creating a New Application" on page 7-11

About Template Support

Before you can create parent and standard tabs, you need to check that your application level template has positions defined for both standard and parent tabs using the appropriate substitution strings. You also need to make sure you do not override this template at the page level.

See Also:

- "About User Interface Templates" on page 7-34 for more information on setting a default page template at the application level
- "About Primary Page Attributes" on page 7-53 for more information on setting a template at the page level

Using Tab Manager to Manage Tab Information

You manage tab information using Tab Manager. You can access Tab Manager from the **Tabs** tab or by clicking **View** from the Page Definition.

To access Tab Manager directly from Application Builder:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Tabs** tab.

Tab Manager appears displaying a graphical representation of the tabs defined in your application.

4. To make another tab current, click the tab.

   Note the two Add buttons. Use the Add button on the upper right side of the graphic to add Parent tabs. Use the Add button at the lower left side of the graphic to add standard tabs.

5. To add a new tab, click **Add** adjacent to the appropriate tab type.

   Think of parent tabs as a container to hold standard tabs. For example, in order to add two levels of tabs you first create a parent tab and then add standard tabs to it.

To access Tab Manager from the Page Definition:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
4. Under Shared Components, click **View** to the right of the Parent Tabs or Standard Tabs heading.

   Tab Manager appears displaying a graphical representation of the tabs defined in your application. The currently selected standard or parent tab is highlighted.

5. To make another tab current, click the tab.
6. To add a new tab, click **Add** adjacent to the appropriate tab type.

About the Standard Tab Tasks List

You can also edit tabs from within Tab Manager by selecting an option from the Standard Tab Tasks list located in the bottom right of the page. For example, to add a new set of standard tabs, select **Create a new tab set**. To add a new standard tab, select **Create new tab**.

Creating a Navigation Bar

Navigation bars (see Figure 8–1) offer an easy way for users to move between pages in an application. The location of a navigation bar depends upon the associated page template. A navigation bar icon enables you to display a link from an image or text. A navigation bar entry can be an image, an image with text beneath it, or text.

You must supply navigation bar entry images and text. When you create a navigation bar entry, you can specify an image, text, a display sequence, or an URL.

Figure 8–1 Navigation Bar Entry

The topics that follow describe how to create a navigation bar entry containing icons and a navigation bar without icons.

Creating a Navigation Bar Entry

Before you can add a navigation bar, you must create a navigation bar entry.

To create a navigation bar entry referencing an icon:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **NavBar** tab.
4. Run the Create NavBar Entry Wizard by clicking **Create**.
5. Specify the following NavBar entry attributes:
   - Sequence
   - Alt Tag Text
   - Icon Image Name
   - Image Height and Image Width
   - Text

Specify the target location.

6. If the target location is an URL:
   - From Target type, select **URL**
   - In URL Target, type an URL
7. If the target location is a page:
   - From Target type, select **Page in this application**
   - In Page, specify the page number
8. If the navigation bar entry will display conditionally, specify the appropriate conditional information and click **Create NavBar Entry**.

To create a navigation bar entry without icons:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **NavBar** tab.
4. Run the Create NavBar Entry Wizard by clicking **Create**.
5. Specify the following icon attributes:
   - Sequence
   - Text
6. Specify the target location:
   - If the target is an URL, use `f?p` syntax to specify the location in the URL Target field. For example:

     ```
     f?p=160:5:&SESSION.
     ```

   - If the target is another page, enter the page number in the Page field.
7. If the navigation bar entry will display conditionally, specify the appropriate conditional information and click **Create NavBar Entry**.

To manage navigation bar information:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Click **View** to the right of the Navigation Bar heading.

   The Navigation Bar page appears.

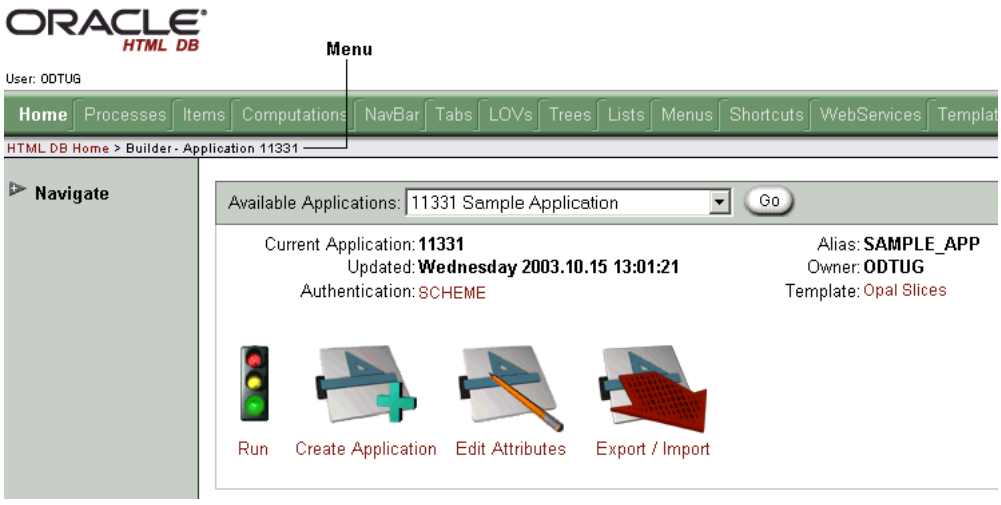
3. On the Navigation Bar page you can:
   - View details about a specific entry by clicking the edit icon
   - Access a grid view, by clicking **Grid View**
   - Create a new icon, by clicking **Create**

Creating Menus

Menus provide users with hierarchical navigation. A menu is a hierarchical list of links that display using templates. You can display menus as a list of links, or as a breadcrumb path.

As shown in Figure 8–2, a breadcrumb style menu indicates where the user is from a hierarchical perspective within the application. In addition, users can click a specific page to instantly view it. You can include a breadcrumb menu that functions as second level of navigation and displays beneath the standard tabs at the top of each page.

Figure 8–2 Breadcrumb Style Menu



Creating a Menu

Before you can add a menu to your application you must create it by running the Menu Wizard.

To create a menu:

1. Click the **Build** icon.
2. When Application Builder appears, select the **Menus** tab.
3. To create a new menu, click **Create** and follow the on-screen instructions.

   Once your menu has been created, you need to add options to it.
4. From the menu list, select the menu you just created and click **Create Menu Options**.
5. On the Menu Options page, specify the following:
   - **Page ID** - Specify the page on which this menu will be current.
   - **Display Sequence** - Indicate the order in which menu options appear.
   - **Parent Menu Option** - Identify the parent of this menu entry.
   - **Short Name** - Specify the short name of this menu option (referenced in the menu template).
   - **Long Name** - Specify the long name of this menu option (referenced in the menu template).
6. Specify a target location.

   If the target location is an URL:
   - From Target type, select **URL**
   - In URL Target, type an URL

   If the target location is a page:
   - From Target type, select **Page in this application**
   - In Page, specify the page number
7. When you are finished defining menu attributes, click **Create**.

Repeat these procedures for each menu option you need to create.

Creating a Menu Template

A menu displays using a template.

To create a menu template:

1. Click the **Build** icon.

2. When Application Builder appears, select the **Templates** tab.
3. Scroll down to Menu Templates, click **Create/Copy**.

   The Create Menu Template Wizard appears.
4. From Create Menu Templates, select whether to create the template from scratch or by copying another template.
5. If prompted, enter the appropriate HTML for your menu using the substitution strings:
   - `#LINK#` - The anchor target of the menu option.
   - `#NAME#` - The short name of the menu option.
   - `#LONG_NAME#` - The long name of the menu option.

Adding a Menu to a Page

Once you create a menu and a menu template, the next step is to add it a page by creating a region and specifying the region type as Menu.

To add a menu to a page:

1. Click the **Build** icon.
2. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
3. Under Regions, click **Create**.
4. While running the Create Region Wizard:
   - Select **Menu** for the region type.
   - Enter a title.
   - Select a menu and menu template.
5. Click **Create Menu Region**.

Repeat these procedures for each page on which you would like to add a menu.

About Creating a Dynamic Menu

To give users a more exact context, you may include session state in a menu, making your menus dynamic. For example, suppose a page in your application displays a list of orders for a particular company and you want to include the following breadcrumb menu:

```
Home > Orders > Orders for ACME Inc
```

In this example, ACME Inc not only indicates the page a user is on, but also the navigation path. The HTML DB engine stores the value of ACME Inc. in session state.

To create this type of dynamic menu, you must include a reference to a session state item in the menu's short name or long name, for example:

```
&COMPANY_NAME.
```

Creating Trees

You can create a tree in your application to effectively communicate hierarchical or multiple level data.

To create a tree:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Trees** tab and click **Create**.
4. Follow the on-screen instructions to enter basic page information.
5. Enter a Tree Name and specify the Default Expanded Levels.
6. Under Start Tree, specify how the starting tree node is created:
   - **Based on new item with popup list of values** - Creates a tree based on a new item with a popup list of values (LOV). Requires an LOV query that selects, displays, and returns values from a tree table. Use this method to select a different starting point each time you visit a page.
   - **Based on a SQL Query** - Creates a tree based on a SQL query. Requires a SQL query that selects a primary key from a tree table.
   - **Static value** - Creates a tree based on a static value.

   Keep in mind that the option **Based on new item with popup list of values** enables you to select a different starting point each time you visit a page. The starting point for the last two options always remains the same.

   You build a tree based on a table that contains the node data. This base table must have an ID (a primary key) and a parent ID that functions as a foreign key of the table. These IDs determine the number of tree levels.

7. Follow the on-screen instructions and specify the owner and name of the table from which the tree will be based.
8. Under Link Option, select **Existing flows item** to make leaf node text a link.

Creating Lists

As shown in Figure 8–3, a list is a shared collection of links. You control the appearance of a list through list templates. Each list element has a display condition which enables you to control when it displays. You can define a list element to be either current or non current for a specific page. You further specify what current looks like using template attributes. You add a list to a page by creating a region and specifying the region type as List.

Figure 8–3 List

Creating a List

Before you can add a list to your application you must create it by running the Lists Wizard.

To create a list:

1. Click the **Build** icon.
2. When Application Builder appears, select the **Lists** tab.
3. To create a new list, click **Create List** and follow the on-screen instructions.

   Once your list has been created, you need to add items to it.

4. Select a list and click **Create List Item**.
5. On the List Item page, specify the Label and Sequence attributes.
6. Specify a target location.

   If the target location is an URL:

   - From Target type, select **URL**
   - In URL Target, type an URL

   If the target location is a page:

   - From Target type, select **Page in this application**
   - In Page, specify the page number
7. When you are finished defining list attributes, click **Create** or **Create and Create Another**.

Adding a List on a Page

Once you created a list, the next step is to add it a page by creating a region and specifying the region type as List.

To add a list to a page:

1. Click the **Build** icon.
2. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
3. Under Regions, click **Create** to run the Create Region Wizard.
4. Select **List** as the region type.
5. Specify region attributes:
   - Enter a title
   - Select a region template
   - Specify a display point
   - Specify a sequence
6. Click **Create List Region**.

Repeat these procedures for each page on which you would like to add a list.

About Creating a List Template

You control the appearance a list through list templates.

To create a new list template:

1. Click the **Build** icon.
2. When Application Builder appears, select the **Templates** tab.
3. Scroll down to List Templates, click **Create/Copy**.

   The Create Menu Template Wizard appears.
4. Specify how to create a new list template and follow the on-screen instructions.

Creating a Branch

A branch is an instruction to link to a specific page, procedure, or URL. For example you can branch from page 1 to page 2 after page 1 is submitted.

You create a new branch by running the Create Page Branch Wizard and specifying Branch Point and Branch Type. The Branch Type defines the type of branch you are creating.

To create a branch:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
4. Under Branching, click **Create** to run the Create Page Branch Wizard.
5. Select a Branch Point:
   - **On Submit: Before Computation** - Occurs before computations, validations, or processing. Use this option for a Cancel button.
   - **On Submit: Before Validation** - Occurs after computations, but before validations or processing. Typically not used. If a validation fails, page processing stops, a rollback is issued, and the page displays the error. Because of this default behavior, you do not need to create branches to accommodate validations. However, you might want to branch based on the result of a computation (for example, to the previous branch point).

- **On Submit: Before Processing** - Occurs after computations and validations, but before processing. Use this option to branch based on a validated session state, but before performing any page processing.
- **On Submit: After Processing** - Occurs after computations, validations, and processing. This option branches to an URL or page after performing computations, validations, and processing. When using this option, remember to sequence your branches if you have multiple branches for a given branch point.
- **On Load: Before Header** - Occurs before a page rendered. This option displays another page instead of the current page or redirects the user to another URL or procedure.

6. Select a Branch Type.

   Depending upon the Branch Type, specify the following types of information on the pages that follows:

   - A page ID of the page you wish to branch to
   - PL/SQL code
   - An URL address

7. Follow the on-screen instructions.

Creating Regions

A region is an area of a page that uses a specific template to generate HTML content. Each page can have any number of regions. You can use regions to group page components (such as items or buttons). You can create simple regions that do not generate additional HTML, or create elaborate regions that frame content within HTML tables or images.

Regions display in sequence within HTML table columns. You can also explicitly place regions in positions defined in the page template. You can also choose to display regions conditionally.

Topics in this section include:

- Creating New Regions
- Building a Form Using a Region
- Building a Report Using a Region
- About Regions Based on an URL

- About Regions Based on PL/SQL Dynamic Content

Creating New Regions

You create new regions by running the Create Region Wizard.

To create a new region:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Under Regions, click **Create**.

   The Create Region Wizard appears.
3. Select a region type and follow the on-screen instructions.

When you create a region you select a region type. The HTML DB engine interprets a region differently based the type you select. Table 8–1 describes available region types.

Table 8–1 Region Types

Region Type	Description
HTML	HTML regions contain HTML you provide. They are also used as containers for items. Any HTML you type may contain substitution strings.
Report	Report regions can be defined by a SQL query you write, or by using a wizard to guide you through the steps needed to write a query. **See Also:** "Creating Reports" on page 8-28
Chart	Chart regions contain line, bar, or pie charts based on SQL queries. **See Also:** "Creating Charts" on page 8-35
List	List regions are used for navigation and may consist of links or images. Individual list entries can be conditionally displayed. **See Also:** "Creating Lists" on page 8-11
Menu	Menu regions are navigational controls consisting of predefined, hierarchically organized links. **See Also:** "Creating Menus" on page 8-7

Table 8–1 Region Types

Region Type	Description
PL/SQL Dynamic Content	Regions based on PL/SQL enable you to render any HTML or text using the PL/SQL Web Toolkit.
Other, HTML Text (escape special characters).	Same as HTML region, but the HTML DB engine escapes special characters before they are rendered.
Other, HTML Text (with shortcuts)	Same as HTML region, but with support for shortcuts. **See Also:** "Utilizing Shortcuts" on page 8-40
Other, Help Text	Help regions enable you to provide page level help. **See Also:** "Creating a Help Page" on page 8-41
Other, URL	URL based regions obtain their content by calling a Web server using a predefined URL. **See Also:** "About Regions Based on an URL" on page 8-17
Other, Web Service Result	Web service based regions call a predefined Web service to obtain an XML document. An optional XSLT style sheet may be applied. **See Also:** "Implementing Web Services" on page 12-15
Other, Tree	Trees are a hierarchical navigational control based on a SQL query executed at runtime. It enables the user to expand an collapse nodes. **See Also:** "Creating Trees" on page 8-10

See Also:

- "Creating Regions" on page 8-14 for more information on creating specific types of regions
- *Oracle Database Application Developer's Guide - Fundamentals* for more information on developing Web applications with PL/SQLs
- *PL/SQL Packages and Types Reference* for more information on htp packages

Building a Form Using a Region

The easiest way to create a region that contains a form is to use the Form on Table or View Wizard. This wizard creates one item for each column in a table. It also

includes the necessary buttons and processes required to insert, update, and delete rows from the table using a primary key. Each region has a defined name and display position all other attributes are items, buttons, processes, and branches.

You can also create a form manually by performing the following steps:

- Create an HTML region (to serve as a container for your page items)
- Create items to display in the region
- Create processes and branches

Building a Report Using a Region

When you define a Report region, you must select one of the following options:

- **Easy Report** - This report does not require any SQL knowledge. You simply select the appropriate schema, table, columns, and result set display.
- **SQL Report** - This report requires some SQL knowledge. When prompted, you enter a SQL query.
- **PL/SQL Function Returning SQL Query** - This report requires some PL/SQL knowledge. When prompted, you enter a function that returns a SQL query.

  This method of defining a report is used when the structure of the query varies based on user input or another session state (for example, if the query is determined at runtime rather than beforehand).

  The following example returns all employees from the `emp` table whose name is similar to the value typed into an page item called *`search_string`*.

```
SELECT ename, job, deptno, sal
FROM   emp
WHERE  upper(ename) LIKE '%'||upper(:search_string)||'%';
```

  Both the *`ename`* column of the *`emp`* table and the page item *`search_string`* are converted into upper case. Using the like operator and appending a percent sign to each side of value of the search_string creates a wild card search

About Regions Based on an URL

Typically, pages in an application are based on data stored in an Oracle database. To incorporate content from other servers, you can use regions based on a URL. For example, suppose you wanted to reference the current Oracle stock price. You could create a region of type URL based on an URL such as the following:

```
http://quote.yahoo.com/q?d=b&s=ORCL
```

You could then create a item called STOCK_SYMBOL and base your region on a stock price entered by the user. For example:

```
http://quote.yahoo.com/q?d=b&s=&STOCK_SYMBOL.
```

Sometimes (as is the case with the previous example) the HTML returned to the region is more than is needed. To restrict the HTML displayed you can use the following region attributes:

- URL (discard until but not including this text)
- URL (discard after and including this text)

Note that the previous example may require that you set the Proxy Server application attribute. If you do not set the Proxy Server application attribute, you will get an error message. Oracle HTML DB uses the Oracle `utl_http.request_pieces` function to obtain the HTML generated from the given URL.

> **See Also:** "Editing Application Attributes" on page 7-29 for more information on setting the Proxy Server application attribute

About Regions Based on PL/SQL Dynamic Content

If you need to generate specific HTML content not handled by Oracle HTML DB forms, reports, and charts, you can use the region type PL/SQL. To generate HTML in this type of region, you need to use the PL/SQL Web Toolkit. You can reference session state using bind variable syntax. Keep in mind that when you generate HTML in this way you do not get the same consistency and control provided with templates.

> **See Also:**
>
> - *Oracle Database Application Developer's Guide - Fundamentals* for more information on developing Web applications with PL/SQL
> - *PL/SQL Packages and Types Reference* for more information on htp packages

To give you more control over HTML dynamically generated within a region, you can use PL/SQL. For example, to print the current date you could create a region with the following source:

```
htp.p(TO_CHAR(SYSDATE,'Day Month DD, YYYY'));
```

This next example accesses tables:

```
DECLARE
   l_max_sal NUMBER;
BEGIN
   SELECT max(sal) INTO l_max_sal FROM emp;
   htp.p('The maximum salary is: '||TO_CHAR(l_max_sal,'999,999.00'));
END;
```

Creating Buttons

As you design your application you can use buttons to direct users to a specific page or URL, or to post or process information (for example, by creating Create, Cancel, Next, Previous, or Delete buttons).

> **See Also:** "About Buttons" on page 7-42 for more information on button naming, branching, and defining other button attributes

Using the Create Button Wizard

You create a button by running the Create Button Wizard from the Page Definition.

To create a new button:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Create an HTML region. (See "Creating New Regions" on page 8-15.)
3. Under Button, click **Create**.

   The Create Button Wizard appears.
4. Select a Task:
   - **Create a button displayed among this region's items**
   - **Create a button in a region position**

   Select **Create a button displayed among this region's items** to add a button to a region as if it was an item (for example, to add a button directly to the right of a form field).
5. Select an Action:

- **Submit page and redirect to URL** - Selecting this action submits the current page to the HTML DB engine whenever a user clicks the button.
- **Redirect to URL** - Selecting this option avoids submitting the page. Choose this action when submitting the page for processing is not necessary (for example, a Cancel button). This action avoids processing in the database and therefore reduces the load.

6. Select a Button Type:
   - **HTML Button (Default)**
   - **Image**
   - **Template Driven**
7. Select **Button is Reset** to create an Undo button. When enabled, this type of button resets the page values to the state they were in when the page was initially rendered.

The sections that follow offer brief descriptions of how to create specific types of buttons.

Creating an HTML Button

Buttons can be placed in predefined region template positions or among items in a form. To create an HTML button, select one of the following while running the Create Button Wizard:

- Under Task, select **Create a button in a region position**
- Under Button Type, select a button type and then **HTML Button (default)**

Creating Lists of Values

A list of values (LOV) is a static or dynamic definition used to display the following types of items:

- Popup lists of values
- Select list
- Check boxes
- Radio groups
- Multiple select lists

Creating LOVs

You define shared (or named) LOVs at the application level by running the LOV Wizard and adding them to the Named List of Values repository. All LOVs can be defined as static or dynamic. Static lists are based on predefined pairs of display and return values. Dynamic lists are based on a SQL query you write that selects values from a table.

To create a named LOV:

1. Click the **Build** icon.
2. When Application Builder appears, select the **LOVs** tab.
3. To create a new LOV, click **Create**.
4. Follow the on-screen instructions.

Referencing Session State within a LOV

You can reference session state by using bind variables. In the following example, this LOV only works if the item called *my_deptno* contains a valid department number.

```
SELECT ename, empno FROM emp WHERE deptno = :my_deptno
```

Inline Static LOV

Static LOVs are based on a static list of display and return values you specify when you run the LOV Wizard. To create a static LOV you run the LOV Wizard and select the LOV type Static. Oracle HTML DB stores the display values, return values, and sort sequence you specify in the Named List of Values repository. Once you add a static LOV to the repository you can create an item and display it as a check box, radio group, select list, or popup lists based on this definition.

You can also use a shorthand syntax to create a static LOV. Simply provide a comma delimited string of values prefaced by the text *STATIC:*. The following example creates a static list containing the options `Yes` and `No`.

```
STATIC:Yes,No
```

In this next example, the LOV would display `Yes` and `No`, but would return `1` and `2`.

```
STATIC:Yes;1,No;0
```

Instead of using a semicolon (;) to delimit values, you can indicate your own delimiters. This is useful approach when the values themselves contain commas. Keep in mind, using your own delimiters can be problematic if your application needs to be translated. The following example uses a tilde (~) and a percent sign (%).

```
STATIC(~,%):Run and Build Flow%RUN_AND_BUILD~Run Flow Only%RUN_ONLY
```

To sort the results in the order they were typed, using the following syntax:

```
STATIC2:1,5,10,20,30
```

Popup LOV

Using a popup LOV is a good choice for lists of values that are too large to return on a single page. Popup LOVs create an icon to the right of a standard text field. When the user clicks this icon a popup window appears with a list of values represented as a series of links. When the user selects from this searchable list, the selected value will be placed in the text field. You control popup LOVs through templates. You can only specify one popup LOV template for each application.

Popup LOVs must be based on a query that selects two columns with different column aliases. For example:

```
SELECT ename name, empno id
   FROM emp
```

Creating Forms

You can include a variety of different types of forms in your applications. You can include forms that enable users to update just a single row in a table or multiple rows at once. Oracle HTML DB includes a number of wizards you can use to create forms automatically, or you can create forms manually.

Topics in this section include:

- Using a Wizard to Build a Form
- Creating a Form Manually
- Processing a Form
- Validating User Input in Forms

Using a Wizard to Build a Form

Oracle HTML DB includes the number of wizards for creating forms. These wizards create complete pages you can later customize.

To create a form using a wizard:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Click the **Wizards** tab.
4. Under Forms, select a wizard.

   Table 8–2 describes the available wizards you can use to create forms.

Table 8–2 Forms Wizards

Wizard	Description
Form on Table or View	Creates a form users can use to update a single row in a database table.
Form on Procedure	Builds a form based on a stored procedure arguments. Use this approach when you have implemented logic or DML (Data Manipulation Language) in a stored procedure or package.
Form on SQL Query	Creates a form based on the columns returned by a SQL query such as an equijoin.
Form on a Table with Report	Creates two pages. One page displays a report. Each row provides a link to the second page to enable users to update each record.
Summary Page	Creates a read-only version of a form. Typically used to provide a confirmation page at the end of a wizard.
Tabular Form	Creates a form in which users can update multiple rows in a database.

5. Follow the on-screen instructions.

Creating a Form Manually

To create a form manually:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Create an HTML region:

- Under Regions, click **Create**
- Select the region type **HTML**
- Follow the on-screen instructions.

3. Start adding items to the page:
   - Under Items, click **Create**
   - Follow the on-screen instructions

Processing a Form

Once you create a form, the next step is to process the data a user types by inserting into or updating the underlying database tables or views. There are three ways to process a form:

- Create one or more processes containing *INSERT*, *UPDATE* and *DELETE* (DML) statements.
- Call an API using a PL/SQL package you create to pass values.
- Use a built-in Automatic Row Processing (DML) process.

Creating an Automatic Row Processing Process

One common way to implement a form is to manually create an Automatic Row Processing (DML) process. This approach offers two primary advantages. First, you are not required to provide any SQL coding. Second, Oracle HTML DB performs DML processing for you.

In order to implement this approach you need to:

- Add items and define the Item Source Type as Database Column and specify a case sensitive column name.
- Select the option **Always overrides the cache value**.

To create an Automatic Row Processing (DML) process:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Under Processes, click **Create**.

   The Create Page Process Wizard appears.
3. In the Name field, type a name to identify the process.

4. In the Sequence field, specify a sequence number.
5. From the Point list, select the appropriate processing point. In most instances, select **Onload - After Header**.
6. From the Type list, select **Automated Row Processing (DML)**.
7. Follow the on-screen instructions.

Creating a Process Containing One or More Insert Statements

In this approach to form handling, you create one or more processes to handle insert, update and delete actions. Instead of having the HTML DB engine handling everything transparently, you are in complete control.

For example, suppose you have a form with three items:

- `P1_ID` - A hidden item to store the primary key of the currently displayed row in a table.
- `P1_FIRST_NAME` - A text field for user input.
- `P1_LAST_NAME` - A text field for user input.

Assume also there are three buttons labeled Insert, Update, and Delete. Also assume you have a table T which contains three columns, `ID`, `FIRST_NAME`, and `LAST_NAME`. The table has a trigger which automatically populates the `ID` column when there is no value supplied.

To process the insert of a new row, you create a conditional process of type PL/SQL that executes when the user clicks the Insert button. For example:

```
BEGIN
    INSERT INTO T ( first_name, last_name )
                VALUES  (:P1_FIRST_NAME, :P1_LAST_NAME);
END;
```

To process the update of a row, you create another conditional process of type PL/SQL. For example:

```
BEGIN
    UPDATE T
           SET first_name = :P1_FIRST_NAME,
                  last_name = :P1_LAST_NAME
     WHERE ID = :P1_ID;
END;
```

To process the deletion of a row, you create a conditional process that executes when the user clicks the Delete button. For example:

```
BEGIN
    DELETE FROM T
    WHERE ID = :P1_ID;
END;
```

Using a PL/SQL API to Process Form Values

For certain types of applications it is appropriate to centralize all access to tables in a single or few PL/SQL packages. If you have created a package to handle DML operations, you can call procedures and functions within this package from a After Submit PL/SQL process to process insert, update and delete requests.

> **See Also:** "Oracle HTML DB APIs" on page 13-1

Populating Forms

Oracle HTML DB populates a form on load, or when the page is being rendered. You can populate a form in the following ways:

- Create a process and define the type as Automated Row Fetch.
- Populate the form manually by referencing a hidden session state item.

To create an Automated Row Fetch process:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Under Processes, click **Create**.

   The Create Page Process Wizard appears.
3. In the Name field, type a name to identify the process.
4. In the Sequence field, specify a sequence number.
5. From the Point list, select the appropriate processing point.
6. From the Type list, select **Automated Row Fetch**.
7. Follow the on-screen instructions.

You can also populate a form manually by referencing a hidden session state item. For example, the following code in an Oracle HTML DB process of type PL/SQL

would set the values of *ename* and *sal*. The example also demonstrates how to manually populate a form by referencing a hidden session state item named P2_ID.

```
FOR C1 in (SELECT ename, sal
FROM emp WHERE ID=:P2_ID)
LOOP
     :P2_ENAME := C1.ename;
     :P2_SAL := C1.sal;
END LOOP;
```

In this example:

- C1 is an implicit cursor
- The value of P2_ID has already been set
- The process point for this process would be set to execute (or fire) on or before **Onload - Before Regions**

Validating User Input in Forms

You can use validations to check data a user types prior to processing. Once you create a validation and the associated error message, you can associate it with a specific item. You can choose to have validation error messages display inline (that is, on the page where the validation is performed) or on a separate error page.

Creating an inline error message involves these steps:

- Create a new validation and specify error message text
- Associate the validation with a specific item

To create a new validation:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Under Validations, click **Create**.
3. When the Create Validations Wizard appears, follow the on-screen instructions.

   Validations Types are divided into two categories:

   - **Item.** These validations start with the phrase "Item" and provide common checks you may want to perform on the item that the validation is associated with.
   - **Code.** These validations require you provide either a piece of PL/SQL code or SQL query that defines the validation logic. Use this type of validation to

perform custom validations that require verifying values of more than one item or accessing additional database tables.

4. Follow the on-screen instructions.

To associate an item with a validation and specify error message text:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Under Validations, select the validation item you want to associate.
3. Scroll down to Error Messaging:
   - In Error message display location, verify the display location
   - In Associated Item, select the item you want to associate with this validation

Creating Reports

In Oracle HTML DB a report is simply the formatted result of a SQL query. You can generate reports by:

- Selecting and running a built-in wizard
- Defining a report region based on a SQL query

> **See Also:** "Building a Report Using a Region" on page 8-17 for more information on defining a report region

Topics in this section include:

- Using a Wizard to Create a Report
- Managing Report Attributes
- Creating a Report with Pagination

Using a Wizard to Create a Report

Oracle HTML DB includes the number of built-in wizards for generating reports.

To create a report using a wizard:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

3. Click the **Wizards** tab.
4. Under Reports, select a wizard.

   Table 8–3 describes the available wizards you can use to create reports.

Table 8–3 Reports Wizards

Wizard	Description
Easy Report	Builds a report page based on the owner, table, columns, and templates you specify. This report does not require any manual SQL coding.
SQL Report	Builds a report page based on a custom SQL statement you provide.
Report with link to form on table (2 pages)	Builds a two page report. The first page enables users to specify the row to be updated. The second page provides users with a form to update or insert the selected row.

5. Follow the on-screen instructions.

Managing Report Attributes

Application Builder offers a great deal of flexibility in defining report attributes. Using the Report Attributes page, you can specify the column display sequence, display report headings, column and heading alignments, column formatting, and sort order. You can further refine attributes of a specific column using the Column Attributes page.

Topics in this section include:

- Accessing Report Attributes
- Enabling Column Sorting
- Exporting a Report
- Creating a Column link
- Defining Updatable Columns
- Defining a Column as a List of Values
- Controlling When Columns Display
- Controlling Column Breaks

Accessing Report Attributes

Using the Report Attributes and Column Attributes pages you can precisely control the look and feel of reports.

To access the Report Attributes page:

1. From the Available Applications list, select an application and click **Go**.
2. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
3. Under Regions, click **Q** next to the name of the report region you want to edit.

   The Report Attributes page appears.
4. Under Report Column Attributes, you can:
   - Click the arrows to change the column display sequence.
   - Under Heading, specify different column headings.
   - Under Column Align., select the column alignment.
   - Under Heading Align., select the heading alignment.
   - Select **Show** to determine whether the column displays.
   - Click **Sum** to enable the sum of a column.
   - Click **Sort** and select a sequence number from Sort Sequence to specify a unique sort sequence.

   You can further refine attributes of a specific column on the Column Attributes page.
5. Under the Report Column Attributes, click the edit icon adjacent to the appropriate column name.

   The Column Attributes page appears, containing the following sections:
   - Column Definition
   - Column Formatting
   - Column Link
   - Updatable Column Attributes
   - List of Values
   - Authorization

- Conditional Display

About Column Definition The Column Definition section of the Column Attributes page contains the same options available on the Report Attributes page. For example, you can specify a column heading, determine whether a column should display in the report, whether to calculate and display a column sum, enable sorting, as well as specify column and column heading alignment.

About Column Formatting You can use the following Column Formatting properties to further customize column display:

- **Number/Date Format** defines a number and date format mask to be applied to a numerical column.
- **CSS Class** defines CSS classes to be applied to a column value.
- **CSS Style** defines a CSS style to be applied to a column value.
- **Highlight Words** specifies text strings to be highlighted in a report column. For multiple highlighted words, use a comma delimited list. Application or page items can be referenced using &ITEM. syntax. (For example, to highlight strings entered into a search field.)
- **HTML Expression** specify an HTML expression to be displayed in the column. Use #COLUMN# syntax to show column values in HTML.

Enabling Column Sorting

You enable column sorting on either Report Attributes or Column Attributes pages.

To enable column sorting on the Report Attributes page:

1. Navigate to the Report Attributes page. (See "Accessing Report Attributes" on page 8-30.)
2. Under Report Column Attributes, click **Sort** adjacent to the column to be sorted.
3. Under Sorting, specify ascending and descending image attributes or click **set defaults**.

Exporting a Report

You can export a report as either a comma delimited file (.csv) or XML file. You specify an export format by selecting a report template.

To specify an export report template:

1. Navigate to the appropriate Report Attributes page. (See "Accessing Report Attributes" on page 8-30.)
2. Under Layout and Pagination, select one of the following from the Report Template list:
   - **export: CSV** exports the report as a CSV file
   - **export: XML** exports the report as a XML file

   Selecting either option prevents the HTML DB engine from rendering the page and dumps the content to either a CSV or XML file.

   You can use the options under CSV Output to create a link that downloads the content of the report.
3. Scroll down CSV Output.
4. To create a link that downloads the content of a report:
   - From the Enable comma separated values (CSV) output list, select **Yes**.
   - In the CSV download link label field, specify link text. This text will display in your report and enable users to invoke a download.

Creating a Column link

Using Column Link attributes, you can create a link from a report to another page.

To create a column link:

1. Navigate to the appropriate Column Attributes page. (See "Accessing Report Attributes" on page 8-30.)

   The Column Attributes page appears.
2. Scroll down to Column Link.
3. In Application, specify the target application ID. To specify the current application, use the following substitution string:

   ```
   &APP_ID.
   ```
4. In Request, specify the request to be used.
5. In Clear Cache, specify the pages (that is, the page ID) on which to clear cache. You can specify multiple pages by listing the page ID in a comma delimited list.
6. Use the Name and Value fields to specify session state for a specific item.

7. In Link Text, enter text to be displayed as a link, specify an image tag, or pick from the list of default images.
8. Click **Generate Link**.

Defining Updatable Columns

You can define how updatable forms display on the Column Attributes page.

To define updatable column attributes:

1. Navigate to the appropriate Column Attributes page. (See "Accessing Report Attributes" on page 8-30.)

   The Column Attributes page appears.
2. Scroll down to Updatable Column Attributes.
3. From Display As, select a type of updatable column.
4. In Width and Height, specify the width and height of the form item.
5. In Element Attribute, define a style or standard form element attribute.
6. In Element Option, specify form element attributes for items in a radio group or check box.

Defining a Column as a List of Values

Report columns may be rendered as lists of values. For example, a column can be rendered using a select list or a popup list of values.

To specify column LOV attributes:

1. Navigate to the appropriate Column Attributes page. (See "Accessing Report Attributes" on page 8-30.)

   The Column Attributes page appears.
2. Scroll down to Updatable Column Attributes.
3. From Named LOV, select a named LOV from the Named List of Values repository.
4. In Display Null, specify whether to include a display null value.
5. In Null Text, specify the value to be returned if Display NULL is selected.
6. If you have not already selected a named LOV, specify a query used to display a select list in LOV Query.

See Also: "Creating Lists of Values" on page 8-20

Controlling When Columns Display

You can use the Authorization and Conditional Display attributes to control when a column displays.

Authorization enables you to control access to resources (such as a report column) based on predefined user privileges. For example, you could create an authorization scheme in which only managers can view a specific report column. Before you can select an authorization scheme, you must first create it.

A condition is a small unit of logic that enables you to control the display of a column based on a predefined condition type. The condition evaluates to true or false based on the values you enter in the Expressions fields.

To specify Authorization and Conditional Display attributes:

1. Navigate to the appropriate Column Attributes page. (See "Accessing Report Attributes" on page 8-30.)
2. From Named LOV, select a named LOV from the Named List of Values repository.
3. Under Authorization, make a selection from the Authorization Scheme list.
4. Under Conditional Display, make a selection from the Type list and depending upon the selected type, enter an expression or value in the appropriate Expression fields.

See Also:

- "Providing Security Through Authorization" on page 10-17
- "Understanding Conditional Rendering and Processing" on page 6-9
- Appendix A, "Available Conditions" on page A-1

Controlling Column Breaks

You can control whether a specific column repeats and how column breaks appear when printed using Break Formatting attributes. For example, suppose your report displays employee information by department number. If multiple employees are members of the same department, you can increase the readability by specifying the department number only appears once.

To create this type of column break:

1. Navigate to the appropriate Report Attributes page. (See "Accessing Report Attributes" on page 8-30.)
2. Scroll down to Break Formatting.
3. Make a selection from the Breaks list.

Creating a Report with Pagination

The HTML DB engine can paginate result sets of a report region in the following ways:

1. Use of conditional buttons and branches, such as Next and Previous buttons to create a custom pagination scheme.
2. Page result sets (that is "Internet style" pagination).
3. Row ranges paginated by set (for example, Row(s) 1-10, 11-20, and 21-23).
4. Row ranges in a self submitted select list (paginated by page).
5. Row ranges paginated by page (for example, 1 - 10 of 23).

You can control options 2, 3, and 4 using report templates, specifically the `#PAGINATION#` substitution string in the After Rows report template attribute.

You can implement options 2, 3, 4, and 5 by configuring report region attributes to enable pagination.

To configure report region attributes to enable pagination:

1. Create a region based on a SQL query. (See "Building a Report Using a Region" on page 8-17.)
2. Under Regions, click **Q** next to the name of the report region you want to edit.

   The Report Attributes page appears.
3. To change pagination, scroll down to Layout and Pagination and select a new Pagination Scheme.
4. To save your changes, click **Apply Changes**.

Creating Charts

Oracle HTML DB includes built-in wizards for generating HTML and SVG (Scalable Vector Graphics) charts.

To create a chart using a built-in wizard:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Click the **Wizards** tab.
4. Under Charts, select a wizard.
   - **HTML Chart** creates a single page containing an HTML horizontal or vertical bar chart.
   - **SVG Chart** creates a SVG (Scalable Vector Graphics) chart.
5. Follow the on-screen instructions.

Creating Calendars

Oracle HTML DB includes a built-in wizard for generating a monthly calendar. Once you specify the table on which the calendar is based you can create drill down links to information stored in specific columns.

To create a calendar using a built-in wizard:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Click the **Wizards** tab.
4. Under Calendar, select **Monthly Calendar**.
5. Follow the on-screen instructions.

Specifying Layout and User Interface

This section describes how to implement common application layout and user interface techniques.

Topics in this section include:

- Creating a Multiple Column Layout
- Using a LOV to Drive Another LOV
- Specifying Print Preview Mode
- Utilizing Shortcuts

Creating a Multiple Column Layout

A region is an area of a page that uses a specific template to format HTML content. You use regions to group page components and items. To create a multiple column layout, you create two regions that display in adjacent cells of the same table.

You can create a multiple column layout by either:

- Manually creating the two adjacent regions
- Defining a page template that contains a multiple column table

Creating Regions in Multiple Columns

You create new regions using the Create Region Wizard. To create a two column page, you create two regions. Oracle HTML DB replaces `#BOX_BODY#` within a two column table and displays the regions in two separate cells.

To create a two column page by creating regions:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
4. Create the first region:
   - Under Regions, click **Create**.

     The Create Region Wizard appears.
   - Select a region type.
   - From the Column field, select **1**.
   - Follow the on-screen instructions.
5. Create the second region:
   - Under Regions, click **Create**.

     The Create Region Wizard appears.
   - Select a region type.
   - From the Column field, select **2**.
   - Follow the on-screen instructions.

Creating a Multiple Column Page Template

Page templates define the appearance of individual pages, including the placement of page components. Each page template is divided into three sections: Header, Body, and Footer. The most basic template must include the substitution string `#BOX_BODY#` in the Body attribute. When the page is rendered, the HTML DB engine replaces `#BOX_BODY#` with HTML to display the regions on that page.

You can create a multiple column page by defining a page template that contains a multiple column table. You then explicitly place regions within specific table cells.

The following example demonstrates how to create a two column page and specify a region position using the `#REGION_POSITION_XX#` substitution string in each column. You would enter this code in the Body section of the page level template.

```
<body #ONLOAD#>
  #FORM_OPEN#
  <table style="width:100%">
    <tr>
      <td style="width:50%;padding:5px;">#REGION_POSITION_01#</td>
      <td style="width:50%; border-left:2px #bbbbbb dashed;
padding:5px;">#REGION_POSITION_02#</td>
    </tr>
  <br />
  #BOX_BODY#
  #FORM_CLOSE#
</body>
```

Once you create this page level template, the newly defined positions would be available as Display Point options when you run the Create Region Wizard.

Using a LOV to Drive Another LOV

You may use a select list to determine the range of values of another select list on the same page. You can achieve this functionality by having a driving select list submit values to a subsequent select list. You incorporate these values in the subsequent select list as a bind variable in the WHERE clause of its query.

You can have one LOV drive another LOV by:

- Creating a basic form.
- Defining two list of values. Note that the driving LOV must submit the page after a value is chosen.
- Defining a branch that branches back to the current page.

See Also:

- "Creating Forms" on page 8-22
- "Creating Lists of Values" on page 8-22
- "Creating a Branch" on page 8-13

Specifying Print Preview Mode

You can optimize a page for printing by creating a specific Print Mode template and specifying that template in the User Template Defaults section of the Edit Application Attributes page. Generally, a Print Mode template optimizes a page for printing. For example, this template might:

- Not display tabs or navigation bars
- Have items display as text instead of form elements

If the UI theme you select does not include a printer friendly template, you can create a Print Mode template by creating a new page template. Although you can create a new page template from scratch, it is generally easier to make a copy of an existing template and then customize it to meet your needs.

To create a new page template:

1. Click the **Build** icon.
2. Select the **Templates** tab.
3. Scroll down to the appropriate template type and click **Create/copy**
4. Follow the on-screen instructions.

See Also: "Working with Templates" on page 7-16

Setting a Print Mode Template for an Application

You enable your Print Mode template by selecting it in User Template Defaults section of the Edit Application Attributes page.

To enable Print Mode mode:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Edit Attributes** icon.

   The Edit Application Attributes page appears.

4. Scroll down to User Interface Templates.
5. From Print Mode Page Template, select your Print Mode template.

Using f?p Syntax to Toggle to Print Mode

Once you create a Print Mode template and select it as an application attribute, you can use `f?p` syntax to toggle to Print Mode. The ninth `f?p` syntax argument (`PrinterFriendly`) enables you turn to this preference on or off. For example, you could include this argument when coding a link, or creating navigation bar icon.

See Also: "Using f?p Syntax to Link Pages" on page 6-20

Utilizing Shortcuts

By using shortcuts you can avoid repetitive coding of HTML or PL/SQL functions. You can use a shortcut to define a component such as a button, HTML text, a PL/SQL procedure, or HTML. Once defined, you can invoke a shortcut using specific syntax unique to the location in which the shortcut is used. Shortcuts can be referenced many times, thus reducing code redundancy.

Defining Shortcuts

Before you can incorporate a shortcut in your application, you must define it and add it to the Shortcuts repository. You reference new shortcuts using the following syntax:

```
"MY_SHORTCUT"
```

Note that the shortcut name must be capitalized and enclosed in quotes.

To define a new shortcut:

1. Click the **Build** icon.
2. Select the **Shortcuts** tab.
3. Click **Create**.
4. Follow the on-screen instructions.

New shortcuts are added to the Shortcut repository and are available for use within the following locations:

- The Region Source attribute of regions defined as HTML (with shortcuts). (See "Creating Regions" on page 8-14)

- Region Header and Footer Text attribute (See "Specifying a Region Header and Footer" on page 7-41)
- Item Label attributes and Item Default Value attribute (See "About Items" on page 7-43)
- Region Templates attributes (See "Editing Templates" on page 7-17)

Creating a Help Page

Oracle HTML DB includes built-in attributes to make creating help for your application quick and easy. Creating help for your application involves the following steps:

- Create a dedicated help page and help region
- Define page help text
- Define item help text
- Create a navigation bar icon to link to your help page

Help created in Oracle HTML DB displays on a dedicated help page. To access help, users click a link that takes them to a dedicated help page. This help page displays page and item help topics specific to the page they are viewing.

Topics in this section include:

- Creating a Help Page and Region
- Defining Help Text
- Creating a Help Navigation Bar Icon

Creating a Help Page and Region

The first step in creating a help for your application it to create a dedicated page and Help Text region.

To create a new help page:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Navigate to a specific page, by either:
   - Entering the page ID in the Edit Page field and click **Go**

- Clicking a page name

4. Under Page in Page Rendering, click **Create**.
5. When prompted to select a region type, click **Other** and then select **Help Text**.
6. Follow the on-screen instructions.

To create a new Help Text region:

1. Navigate to the Page Definition of your help page.
2. Under Regions, **Create**.
3. When prompted to select a region type, click **Other** and then select **Help Text**.
4. Follow the on-screen instructions.

Defining Help Text

You define page and item help text as an attribute of the page and item. Ideally, you would define these attributes as you create your application. For simplicity, however, the following procedures describe how to define this text after the fact.

To define page help text:

1. Navigate to the Page Definition for the page for which you want to add page help.
2. Under Page, click **Edit** to view the existing page attributes.
3. Scroll down to **Page Help Text**.
4. Enter your help text in the field provided.
5. Click **Apply Changes**.

Repeat the previous procedure for each page requiring page help text.

To define item help text:

1. Navigate to the Page Definition for the page for which you want to add item help.
2. Under Items, click name of the item you want to edit.
3. Scroll down to **Help Text**.
4. Enter your help text in the field provided.
5. Click **Apply Change**.

Repeat the previous procedure for each item requiring help text.

Creating a Help Navigation Bar Icon

Once you have created your help, the next step is to create a navigation bar icon so users can link to it.

To create a navigation bar icon:

1. Navigate to the Page Definition.
2. Under Navigation Bar, click **Create**.
3. Specify the appropriate NavBar entry attributes:
   - Sequence
   - Alt Tag Text
   - Icon Image Name
   - Image Height and Image Width
   - Text

   Specify the target location.
4. To specify the target location:
   - From Target type, select **Page in this application**.
   - In Page, specify the page number.
   - In Request, type:

     ```
     &APP_PAGE_ID.
     ```

     By specifying substitution string *&APP_PAGE_ID* as the Request, you are instructing the HTML DB engine to display help text for the current page when the user clicks this icon.

Sending E-mail from an Application

You can send an e-mail from an Oracle HTML DB application by calling a PL/SQL package called `HTMLDB_MAIL`. This package is built on top of the Oracle supplied `UTL_SMTP` package. Because of this dependence, in order to use `HTMLDB_MAIL`, the `UTL_SMTP` package must be installed and functioning.

> **See Also:** *PL/SQL Packages and Types Reference* for more information about the UTL_SMTP package

The most efficient approach to sending mail is to create a background job (or DBMS_JOB package) to periodically send all mail messages stored in the active mail queue. DBMS_JOB package is automatically created when you install Oracle HTML DB. This package pushes the mail queue every 15 minutes. DBMS_JOB package has two parameters: The default p_smtp_host is localhost and the default p_smpt_portno is 25 on install.

- p_smtp_portno is the hostname of your SMTP gateway. The default value is localhost.
- p_smtp_host is the port number of your SMTP gatway. The default value is 25.

> **See Also:** "Managing Engine Settings" on page 15-5 for more information on changing these default values

To enable a user to send an outbound e-mail message from your application, you invoke the HTMLDB_MAIL.SEND procedure.

Oracle HTML DB stores unsent e-mail messages in a table named HTMLDB_MAIL_QUEUE. You can deliver mail messages stored in this queue to the specified SMTP gateway by invoking the procedure HTMLDB_MAIL.PUSH_QUEUE. This procedures requires two input parameters:

- p_smtp_hostname defines the hostname of your SMTP gateway
- p_smtp_portno defines port number of your SMTP gateway (for example, 25)

Oracle HTML DB logs successfully submitted message in the table HTMLDB_MAIL_LOG with the timestamp reflecting your server's local time.

The most efficient approach to sending mail is to create a background job (or DBMS_JOB package) to periodically send all mail messages stored in the active mail queue.

> **See Also:** *Oracle Database Administrator's Guide* for more information managing job queues

The following example demonstrates the use of the HTMLDB_MAIL.PUSH_QUEUE procedure using a shell script. This example only applies to UNIX/LINUX installations. In this example, the SMTP gateway hostname is defined as smtp01.oracle.com and the SMTP gateway port number is *25*.

```
SQLPLUS / <<EOF
EXEC FLOWS.HTMLDB_MAIL.PUSH_QUEUE('smtp01.oracle.com','25');
DISCONNECT
EXIT
EOF
```

9

Debugging an Application

This section describes a number of approaches to debugging your application including viewing Debug Mode, enabling SQL tracing, viewing page reports, and how to manually remove a component to isolate a problem.

This section contains the following topics:

About Tuning Performance

For applications having a large number of concurrent users, maintaining optimal performance is critical. To optimize your application's performance, remember to utilize the following Oracle HTML DB features:

- Use bind variables within your application whenever possible. You can reference session state values using bind variable syntax in SQL queries and application logic such as processes and validations. Accessing session state using bind variables is the most efficient way to reference session state.

- Include a `#TIMING#` substitution string in the region footer so that you can view the timing of each region.

See Also:

- "About Bind Variables" on page 6-18
- "Using Substitution Strings" on page 6-22

Remembering to Review Session State

Many application are based on data contained within application items. For example, buttons may display conditionally based on a value stored in session state. You can view current session state for a page in your application by clicking the Session link on the Developer Toolbar.

See Also:

- "Using the Developer Toolbar" on page 7-10
- "Viewing Session State" on page 6-13
- "Managing Session State Values" on page 6-14
- "Administering Session State and User Preferences" on page 11-4

Accessing Debug Mode

Viewing a page in Debug Mode is effective way to track what the HTML DB engine is doing as it renders a page. You access Debug mode by clicking the **Debug** link in the Developer Toolbar.

See Also: "Using the Developer Toolbar" on page 7-10

Debug Mode displays time codes that correspond to specific HTML DB engine functions. This can be useful if you want to determine when the engine is setting session state. The bottom of the page displays an augmented version of the Page Definition. In addition to enabling you to link to page and component attributes, you can view additional details about item names and computation and processing points. To exit Debug mode, click **No Debug** in the Developer Toolbar.

You can also use `f?p` syntax run a application in Debug mode. Simply call the page and set the Debug argument to `YES`. For example:

```
f?p=100:1:&SESSION::YES
```

See Also: "Using f?p Syntax to Link Pages" on page 6-20

Enabling SQL Tracing and Using TKPROF

Tracing your session can be a very effective way to debug an application. From a database perspective, each page request is a single database session. If you enable SQL tracing, then Oracle HTML DB creates a temporary file you can then analyze using the TKPROF utility.

You enable SQL tracing in Oracle HTML DB by using `f?p` syntax to set the argument `p_trace=YES`. For example, to trace the display of page 1 in application 100 you would use the syntax:

```
http:/.../f?p=100:1&p_trace=YES
```

To use the TKPROF utility:

- Navigate to the directory in which the trace file is created.
- Type the following to view instructions about using TKPROF utility:

  ```
  tkprof help=yes
  ```

> **See Also:** *Oracle Database Performance Tuning Guide* for more information on using the TKPROF program or contact your database administrator

Monitoring Application and Page Resource Use

You can monitor the resource use of Oracle HTML DB applications and pages by calling the built-in package `DBMS_APPLICATION_INFO`. Whenever the HTML DB engine renders or processes a page, the module is set to `HTML DB` and includes the application ID and page ID. Once set, you can use the `V$SESSION` and `V$SQLAREA` views to monitor transactions.

Viewing Page Reports

Every Page Definition includes a Page Reports list in the left navigation pane. Each report offers a different presentation of the components that define the page. In

particular, the Page Detail Report describes all currently defined page components and processes.

To access the Page Detail Report:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Navigate to the appropriate Page Definition.
4. Click **Page Reports** in the left navigation pane to expand the list.
5. Click **Page Detail (popup page)**.

   Once the Page Detail Report appears, you can select the following options at the top of the page to display or hide information:

   - **All.** Enabled by default. Displays detailed information about the current page including defined regions, items, buttons, processes, validations, branches, and computations.
   - **Regions, Items, and Buttons.** Displays detailed information of all items and buttons defined in each region of the page.
   - **Processes.** Displays details about defined processes including source code.
   - **Validations.** Displays any defined validations performed on the current page.
   - **Branches.** Displays information about branching performed on the current page.
   - **Computations.** Displays details about computations on the current page defined at the page or application level.

   See Also: "Viewing Page Reports" on page 7-7

Debugging Problematic SQL Queries

If your query does not seem to be running correctly, try running it in SQL Plus or in SQL Workshop. Either approach will test your query outside of the context of your application, making it easier to determine what the problem is.

Removing Components to Isolate a Problem

If you have problems running a page, try removing components one at time. Using this approach, you can quickly determine which components may be the source of your problem. You can quickly disable a component by selecting the conditional display attribute NEVER.

To remove a component using conditional display attributes:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Navigate to the appropriate Page Definition and then to the attributes page for the component you wish to disable.
4. Scroll down to the Conditional Display attribute and select **NEVER**.
5. Click **Apply Changes** and return to the Page Definition.
6. Try running the page again.
7. Continue to remove components until the page runs correctly.

See Also:

- "Viewing a Page Definition" on page 7-5
- "Viewing Page Attributes" on page 7-37
- "Understanding Conditional Rendering and Processing" on page 6-9
- "Running a Page" on page 7-57

10

Managing an Application

This section provides information about Application Builder utilities, how to export and import an application, and how to manage application security.

This section contains the following topics:

Accessing Application Builder Utilities

Application Builder includes a number of utilities to help you manage your application. You can access these utilities from the Application Builder Utilities tab.

To access Application Builder utilities:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Utilities** tab.

   The Application Builder Utilities page appears displaying the following utilities:

   - Translate Application
   - Manage CSS and Image Files

- Export/Import
- View Export Repository

See Also:

- About Translating an Application and Globalization Support on page 16-1
- "Exporting and Importing Applications" on page 10-2

Viewing Application Summary and Utilization Reports

The bottom of the Application Builder Utilities page displays the following two lists:

- Application Summary Reports
- Utilization Reports

To view a specific report:

1. Click the appropriate list to expand it.
2. Select a report title.
3. Follow the on-screen instructions.

Be aware that you can also access application reports by making selections in the Application Navigation Pane.

See Also:

- "About the Application Navigation Pane" on page 7-4
- "Viewing Page Reports" on page 7-7

Exporting and Importing Applications

To move an application from one instance of Oracle HTML DB to another, you must export the application definition to a file. Exporting your application definition is the first step toward deploying it outside of your development environment.

You export and import application definitions and all associated files using the following utilities:

- **Export.** If you are an administrator, you can use Export to export an application, a workspace (Workspace administrator's only), and other related files such as cascading style sheets, images, static files, and script files.

- **View Export Repository.** Use View Export Repository to store and install export files imported into Oracle HTML DB. Once imported, you can view imported files in this repository and then install them into your workspace.

Topics in this section include:

How Exporting an Application Works

Whether you are moving an application to another workspace or just making a copy of it, the export process involves the following steps:

- Export the application and all related files using Export.
- Import the exported files into the target Oracle HTML DB instance. Note that if the target instance is a different database, you also need to export and import any required database objects.
- Install the exported files from View Export Repository

You can import an application into your workspace regardless of the workspace in which it was developed.

About Managing Database Objects

Before you export an application and the appropriate related files, you need to determine if you also need to migrate the database objects referenced by the application.

If the target Oracle HTML DB instance is different from the development environment, you will need to migrate the database objects referenced by the application. In many cases this process can be as simple as using Oracle database export and import utilities to copy the application schema from the development environment to target Oracle HTML DB instance. The following are two common scenarios where this approach would not work:

- When the object development schema refers to tablespaces to which the target instance schema does not have access

- When the development instance schema has sample data that you do not to want migrate to the target instance schema

If a database administrator or an Oracle HTML DB administrator is the person responsible for exporting Oracle HTML DB applications, be sure to clearly communicate if he or she:

- Should include all data when exporting your application
- Should not include data from specific tables you identify

Exporting an Application and Related Files

Use Export to export an application and all related files. When you export a application, Oracle HTML DB generates a text file containing PL/SQL calls.

To export an application:

1. Click the **Build** icon and select the **Utilities** tab.

   The Application Builder Utilities page appears.

2. Click **Export Import**.
3. When prompted to select a task, select **Export** and click **Next**.

   Export appears. To view a list of existing applications, expand the **Existing Applications** list at the bottom of the page.

4. From the Application list, select an application.
5. From File Format, select how rows in the export file will be formatted by choosing one of the following:
   - **UNIX.** Results in a file containing rows delimited by line feeds.
   - **DOS.** Results in a file containing rows delimited by carriage returns and line feeds.
6. From Owner Override, select an optional overriding owner for this application.

   Entries made here replace the Owner attribute when the application is imported into the target instance. As discussed in "About Application Definition" on page 7-30, the Owner attribute indicates the schema against which all of the application's SQL and PL/SQL will be parsed. This feature is useful when the development version of an application runs against a differently named schema then the one in the target instance.

7. From Build Status Override, select one of the following:

- **Run Application Only**
- **Run and Build Application**

Only select **Run Application Only** if you wish to run the application in the target instance but make it inaccessible to developers.

8. Use **As of** to export your application as it was previously defined. Specify the number of minutes in the field provided.

   This utility uses the `DBMS_FLASHBACK` package. Because the timestamp to System Change Number (SCN) mapping is refreshed approximately every five minutes, you may have to wait that amount of time to locate the version you are looking for. The time undo information is retained and influenced by the startup parameter `UNDO_RETENTION` (the default is three hours). However, this only influences the size of the undo tablespace. While two databases may have the same `UNDO_RETENTION` parameter, you will be able to go back further in time on a database with fewer transactions since it is not filling the undo tablespace, forcing older data to be archived.

9. Click **Export Application**.

Exporting Related Application Files

In addition to exporting the actual application file, you also use Export to export other related files such cascading style sheets, images, and script files.

To export related application files:

1. Click the **Build** icon and then select the **Utilities** tab.

   The Application Builder Utilities page appears.

2. Click **Export Import**.

3. When prompted to select a task, select **Export** and click **Next**.

4. To export cascading style sheets, images, files, and script files, select one of the following tabs.

   - CSS
   - Images
   - Files
   - Script Files

Note that when you choose to export cascading style sheets, images, files, or script files, this utility exports all files of the selected type in the workspace. Be aware that exporting workspace images *only* exports those images in your repository that are not associated with a specific application. If all of your images are associated with specific applications then the workspace image export file will be empty.

5. Follow the on-screen instructions

> **See Also:** "Exporting and Importing Applications" on page 10-2 for more information on using the Export Workspace tab

Importing Exported Application Files

Once you export an application and any related files, you need to import them into the target Oracle HTML DB instance before you can install them.

To import an application and related files:

1. Click the **Build** icon and then select the **Utilities** tab.

   The Application Builder Utilities page appears.

2. Click **Export Import**.
3. When prompted to select a task, select **Import** and click **Next**.
4. In Import file, specify the file you are importing.
5. From File Type, select the type of file you are importing and click **Next**.

   Once you have imported a file, you have the option to install it. You can also install it later from the View Export Repository.

> **See Also:** Installing Files from the View Export Repository on page 10-6

Installing Files from the View Export Repository

Once you have imported files into the target Oracle HTML DB instance, you must install them before they become active in Application Builder.

To install files stored in the View Export Repository:

1. Click the **Build** icon and then select the **Utilities** tab.

   The Application Builder Utilities page appears.

2. Click **View Export Repository**.
3. To install a file, select it and click **Install** adjacent to the file you wish to install.

In addition to installing files, you can also use this page to:

- Import an application and related files into the View Export Repository, by clicking **Import File**.
- Delete an export file by selecting it and clicking **Delete File**.

To import an application and related files into the View Export Repository:

1. Click the **Build** icon and then select the **Utilities** tab.

   The Application Builder Utilities page appears.
2. Click **Import File**.
3. Follow the on-screen instructions.

   Once you have imported your application and the related files into the View Export Repository, you have the option of installing it.
4. To install your application, click **Install**.

   The Install Application page appears.
5. From Parse as Schema, select a schema. This is the schema against which all of the application's SQL and PL/SQL will be parsed
6. From Build Status, select one of the following:
   - **Run Application Only**
   - **Run and Build Application**

   Only select **Run Application Only** if you wish to run the application in the target instance but make it inaccessible to developers.
7. From Install As Application, select one of the following:
   - **Reuse Application ID from Export File**
   - **Auto Assign New Application ID**
   - **Change Application ID**

   Use these options to avoid application ID conflicts. These options come in handy when you need to have two versions of the same application in the same workspace. For example, if you are migrating an application to a production instance but still need to maintain development version.

Uploading CSS, Images, and Static Files

You can upload cascading style sheets, images, and static files to your workspace using the CSS Repository, Image Repository, and Static File Repository.

To upload cascading style sheets, images, and static files to your workspace:

1. Click the **Build** icon and then select the **Utilities** tab.
2. Click **Manage CSS and Image Files**.

   The CSS Repository appears.
3. Select the appropriate tab:
   - **Cascading Style Sheets (CSS)**
   - **Images**
   - **Static Files**
4. Click **Create**.
5. Follow the on-screen instructions.

Understanding Security

You can provide security for your application through authentication and authorization. **Authentication** is the process of establishing users' identities before they can access an application. **Authorization** controls user access to specific components based on predefined user privileges. You create and manage both authentication and authorization on the Security Home page.

To access the Security Home page:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Security** tab.
4. When the Security Home page appears, select either **Authentication** or **Authorization**.

See Also:

- "Establishing User Identity Through Authentication" on page 10-9
- "Providing Security Through Authorization" on page 10-17

Using the Security Navigation Pane

The Security Navigation pane displays on the left side of the Security Home page and offers quick access to a number a security functions.

To access the Security Navigation pane:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Security** tab.

   The Security Navigation pane displays on the left side of the page.
4. Click **Navigate** to expand the list.

Establishing User Identity Through Authentication

Authentication is the process of establishing each user's identify before they can access your application. Authentication may require a user identify a username and password or could involve the use of digital certificates or a secure key.

When you create an authentication scheme, you have the option of choosing from a number of preconfigured authentication schemes, copying an authentication scheme from an existing application, or creating your own custom authentication scheme.

Topics in this section include:

- Understanding How Authentication Works
- Creating an Authentication Scheme
- Using the Authentication Scheme Repository
- Viewing the Current Authentication Scheme for an Application
- About Preconfigured Authentication Schemes
- About Creating an Authentication Scheme from Scratch

Understanding How Authentication Works

You determine how your application interacts with users. If all users have the same rights and privileges they are referred to as public users. However, if your application needs to track each user individually, you need to specify an authentication method.

Authentication establishes the identity of each user who accesses your application. Many authentication processes require a user provide some type of credentials such as a username and password. These credentials are then evaluated and they either pass or fail. If the credentials pass, the user has access to the application. Otherwise, access is denied.

Once a user has been identified, the HTML DB engine keeps track of each user by setting the value of a built-in substitution string. As a user navigates from page to page, the HTML DB engine sets the value of `APP_USER` to identify who they are. The HTML DB engine uses `APP_USER` as one component of a key for tracking each user's session state.

From a programming perspective, you can access `APP_USER` using the following syntax:

- From PL/SQL:

  ```
  v('APP_USER')
  ```

- As a bind variable from either PL/SQL or SQL:

  ```
  :APP_USER
  ```

You can use `APP_USER` to perform your own security checks and conditional processing. For example, suppose you created the following table:

```
CREATE TABLE my_security_table (
  user_id   VARCHAR2(30),
  privilege VARCHAR2(30));
```

Once created, you could populate this table with user privilege information and then use it to control the display of application pages, tabs, navigation bars, buttons, regions, or any other application component.

See Also: "Using Substitution Strings" on page 6-22

Creating an Authentication Scheme

As you create your application, you need to determine whether to include authentication. You can:

- **Choose to not require authentication.**

  Oracle HTML DB does not check any user credentials. All pages of your application are accessible to all users.

- **Select a built-in authentication scheme.**

Create an authentication method based on available preconfigured authentication schemes. Depending on which scheme you choose, you may also have to configure the corresponding components of Oracle 9*i*AS, Oracle Internet Directory, or other external services.

- **Create custom authentication scheme.**

  Create a custom authentication method, giving you complete control over the authentication interface. To implement this approach, you must provide a PL/SQL function the HTML DB engine executes before processing each page request. This function's Boolean return value determines whether the HTML DB engine processes the page normally or displays a failure page.

To create an authentication scheme:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Security** tab.
4. Select **Authentication**.

   The Authentication Schemes page appears.
5. To create a new authentication scheme, click **Create Scheme**.
6. Specify how the scheme should be created by selecting one of the following:
   - **Based on preconfigured scheme**
   - **As a copy of an existing scheme**
   - **From Scratch**
7. Follow the on-screen instructions

See Also:

- "About Preconfigured Authentication Schemes" on page 10-12
- "About Creating an Authentication Scheme from Scratch" on page 10-15

Using the Authentication Scheme Repository

Once created, available authentication schemes display in the Authentication Schemes Repository.

To navigate to the Authentication Schemes Repository:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Security** tab.
4. Select **Authentication**.

From the Authentication Schemes Repository, you can:

- Make an authentication scheme current by selecting the **make current** link
- Edit an authentication scheme by clicking the edit icon
- View a flow chart explanation of an authentication scheme by clicking the View icon
- Create a new authentication scheme by clicking **Create Scheme** and following the on-screen instructions

Viewing the Current Authentication Scheme for an Application

To view the current authentication scheme for an application:

1. Click the **Build** icon.

   The list of pages for the selected application appears at the bottom of the page.
2. Select the **Edit Attributes** icon.

   The Edit Application Attributes page appears.
3. Scroll down to the Session Management attribute and click **manage**.

   The Authentication Schemes page appears. Available authentication schemes display in the Authentication Schemes Repository. You apply an authentication scheme to an application by designating it as current.
4. To apply an authentication scheme to the current application, select the **make current** link

About Preconfigured Authentication Schemes

When you select a preconfigured authentication scheme, Oracle HTML DB creates an authentication scheme for your application that follows a standard behavior for authentication and session management. The following list describes available preconfigured authentication schemes:

- **Open Door Credentials** enables anyone to access your application using a built-in login page which captures a username. This can be useful during application development.
- **HTML DB Account Credentials** refers to the internal user accounts (also known as "cookie user" accounts) created and managed in the Oracle HTML DB user repository. Using this scheme authentication method, your application can easily authenticate against these accounts.
- **LDAP Credentials Verification** requires you specify configuration parameters about the external LDAP directory you will be using.
- **No Authentication (using DAD)** gets the username from the Database Access Descriptor (DAD), either as the value stored in the DAD configuration or, if the account information is not stored in the DAD configuration, as the username captured using the basic authentication challenge.
- **Oracle 9iAS Single Sign-On** (HTML DB engine as Partner App) delegates authentication to the 9iAS Single Sign-On (SSO) Server. To you use authentication scheme, your site must have already been registered as a partner application with the SSO server. For more information, contact your administrator.
- **Oracle 9iAS Single Sign-On** (My application as Partner App) delegates authentication to the SSO server. In this case, you must register an application with SSO as a partner application. See the next page for more details.

About DAD Credentials Verification

Database Access Descriptor (DAD) database authentication uses the Oracle database native authentication and user mechanisms to authenticate users using a basic authentication scheme. To use DAD credentials verification:

- Each application user must have a user account in the Oracle database.
- You must configure a PL/SQL DAD for basic authentication (without account information).

  This results in one username/password challenge for browser session for your application users. The user identity token is then made available in the `APP_USER` item.

DAD database authentication is useful when you need to implement an authentication method that requires minimal setup for a manageable number of users. Ideally these users would already have self-managed accounts in the

database and your use of this authentication method would be short lived (for example, during the demonstration or prototyping stages of development).

The main drawback of this approach is burdensome account maintenance, especially if users do not administer their own passwords, or if their database accounts exist only to facilitate authentication to your application.

About HTML DB Account Credentials

HTML DB Account Credentials authentication uses internal user accounts (also known as "cookie user" accounts) created and managed in the Oracle HTML DB user repository. Workspace administrators can create and edit user accounts using the Manage Users page. HTML DB Account Credentials is a good solution when:

- You want control of the user account repository
- Username and password based approach to security is sufficient
- You do not need to integrate into a single sign-on framework

This is an especially good approach when you need to get a group of users up and running on a new application quickly.

> **See Also:** "Managing Users" on page 11-2 for more information on creating and managing user accounts

About LDAP Credentials Verification

Any authentication scheme that uses a login page may be configured to use Lightweight Directory Access Protocol (LDAP) to verify the username and password submitted on the login page. Application Builder includes wizards and edit pages that explain how to configure this option. These wizards assume that an LDAP directory accessible to your application for this purpose already exists and that it can respond to a `SIMPLE_BIND_S` call for credentials verification. When you create a LDAP Credentials authentication scheme, the wizard requests and saves the LDAP host name, LDAP port, and the DN string. An optional pre-processing function can be specified to adjust formatting of the username passed to the API.

About Single Sign-On Server Verification

Oracle HTML DB applications can operate as partner applications with Oracle Application Server's Single Sign-On (SSO) infrastructure. To accomplish this, you must register your application (or register the HTML DB engine) as the partner application. To register your application or the HTML DB engine as a partner

application, follow the Oracle Application Server instructions for registering partner applications and install the Oracle9iAS Portal Developer Kit (PDK).

If you choose this approach, your application will not use an integrated login page. Instead, when a user accesses your application in a new browser session, the HTML DB engine redirects to the Single Sign-On login page. After the user is authentication by SSO, the SSO components redirect back to your application, passing the user identity and other information to the HTML DB engine. The user can then continue to use the application until they log off, terminate their browser session, or until some other session-terminating event occurs.

About Creating an Authentication Scheme from Scratch

Creating an authentication scheme from scratch gives you complete control over your authentication interface. This is the best approach for applications when any of the following is true:

- Database authentication, or other methods are not adequate.
- You want to develop your own login form and associated methods.
- You want to delegate all aspects of user authentication to external services such as Oracle 9*i*AS Single Sign-On.
- You want to control security aspects of Oracle HTML DB session management.
- You want to record or audit activity at the user or session level.
- You want to enforce session activity or expiry limits.
- You want to program conditional n-way redirection logic before Oracle HTML DB page processing.
- You want to integrate your application with non-Oracle HTML DB applications using a common session management framework.
- Your application consists of multiple applications that operate seamlessly (for example, more than one Oracle HTML DB application ID).

> **See Also:** "HTMLDB_CUSTOM_AUTH" on page 13-42 for more information

About Session Management Security

When running custom authentication, Oracle HTML DB attempts to prevent two improper situations:

- Intentional attempts by a user to access session state belonging to someone else. However, users can still type in an arbitrary application session ID into the URL.
- Inadvertent access to a stale session state (probably belonging to the same user from an earlier time). This would commonly result from using bookmarks to application pages.

Oracle HTML DB checks that the user identity token set by the custom authentication function matches the user identity recorded when the application session was first created. If the user has not yet been authenticated and the user identity is not yet known, the session state being accessed does not belong to someone else. These checks determine whether the session ID in the request can be used. If not, the HTML DB engine redirects back the same page using an appropriate session ID.

Building a Login Page

When you create a new application in Oracle HTML DB, a login page is created. The alias for the page is `'LOGIN'`. You can use this page as the 'invalid session page' in an authentication scheme. The page is constructed with processes that call the Oracle HTML DB login API to perform credentials verification and session registration.

You can also create a login page after you create your application by selecting the **Extend this Application** link from the Tasks list on the Application Builder home page. You can also build your own login pages using the pre-built pages as models and tailoring all of the UI and processing logic to your requirements.

To create a login page after you create your application:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. From the Tasks list on the right side of the page, select **Extend this Application**.
4. Select **A login page**, click **Next**, and follow the on-screen instructions.

About Deep Linking

Deep linking refers to the ability to link to an Oracle HTML DB page out of context (for example, from a hyperlink in an e-mail or workflow notification). When you link to a page out of context and the application requires the user be authenticated, the user will be taken to the login page. After credentials verification, the HTML DB

engine automatically displays the page that was referenced in the original link. Deep linking support is supported for applications that use authentication schemes.

Providing Security Through Authorization

Authorization is a broad term for controlling access to resources based on predefined user privileges. While conditions control the rendering and processing of specific page components, authorizations control user access to specific components.

Topics in this section include:

- How Authorization Schemes Work
- Creating an Authorization Scheme
- Attaching an Authorization Scheme to an Application, Page, or Component
- Viewing the Authorization Scheme Utilization Report

How Authorization Schemes Work

An authorization scheme extends the security of your application's authentication scheme. You can specify an authorization scheme for an entire application, a page, or specific component such as a region, item, or button. For example, you could use an authorization scheme to selectively determine which tabs, regions, or navigation bars a user sees.

An authorization scheme either succeeds or fails. If a component level authorization scheme succeeds, the user can view the component. If it fails, the user cannot view the component. If an application or page level authorization scheme fails, then Oracle HTML DB displays a previously defined message.

When you define an authorization scheme you give it a unique name. Once defined, you can attach it to any component in your application. To attach an authorization scheme to a component in your application, simply navigate to the appropriate attributes page and select an authorization scheme from the Authorization Scheme list.

Creating an Authorization Scheme

Before you can attach an authorization scheme to an application or an application component, you must first create it.

To create an authorization scheme:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Security** tab.
4. When the Security Home page appears, select **Authorization**.
5. Click **Create**.
6. Specify how to create an authorization scheme by selecting one of the following:
   - From Scratch
   - As a Copy of an Existing Authorization Scheme
7. Follow the on-screen instructions.

To edit attributes of an existing authorization scheme:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Security** tab.
4. When the Security Home page appears, select **Authorization**.

   Existing Authorization Schemes display at the bottom of the page.
5. To edit attributes for an existing authorizations scheme, click the edit icon.
6. Follow the on-screen instructions.

About the Evaluation Point Attribute

You can specify when your authorization scheme is validated in the Evaluation Point attribute. You can choose to have your authorization scheme validated once for each session or once for each page view.

Keep in mind, that if you specify that an authorization scheme should be evaluated once for each session and the authorization scheme passes, the underlying code, test, or query will not be executed again for the duration of the application session. If your authorization scheme consists of a test whose results might change if evaluated at different times during the session, then you should specify that the evaluation point be once for each page view.

About Resetting Authorization Scheme State

If an authorization scheme is validated once for each session, Oracle HTML DB caches the validation results in each user's session cache. You can reset a session's

authorization scheme state by calling the `HTMLDB_UTIL.RESET_AUTHORIZATIONS` API:

Calling this procedure nulls out any previously cached authorization scheme results for the current session. Be aware that this procedure takes no arguments and is part of the publicly executable `HTMLDB_UTIL` package.

See Also: "RESET_AUTHORIZATIONS Procedure" on page 13-9

Attaching an Authorization Scheme to an Application, Page, or Component

Once you have created an authorization scheme you can attach it to an entire application, page, or component.

To attach an authorization scheme to an application:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Edit Attributes** icon.

   The Edit Application Attributes page appears.
4. Scroll down to Authorization and make a selection from the Authorization Scheme list.

To attach an authorization scheme to a page:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

   The list of pages for the selected application appears at the bottom of the page.
3. Navigate to a specific page, by either:
   - Entering the page ID in the Edit Page field and clicking **Go**
   - Clicking the page name
4. Under Page, click **Edit** to view the page attributes.
5. Scroll down to Security and make a selection from the Authorization Scheme list.

To attach an authorization scheme to a page component:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.

The list of pages for the selected application appears at the bottom of the page.

3. Navigate to a specific page, by either:
   - Entering the page ID in the Edit Page field and clicking **Go**
   - Clicking the page name
4. Access the attributes for the component to which you want to apply the authorization scheme. (See "Managing Page Rendering Components" on page 7-39.)
5. Scroll down to the Authorization attribute and make a selection from the Authorization Scheme list.

Viewing the Authorization Scheme Utilization Report

You can use the Authorization Scheme Utilization Report to view details about authorization schemes included in your application.

To view Authorization Scheme Utilization Report:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select **Security** tab.
4. When the Security Home page appears, select **Authorization**.

   The Security Navigation pane displays on the left side of the page.
5. Click **Navigate** to expand or collapse the list.
6. Select **Utilization**.
7. Make a selection from the Authorization Scheme list and click **Go**.

11

Managing Your Development Workspace

In the Oracle HTML DB development environment, developers log in to a shared work area called a workspace. Users are divided into two primary roles: *developer* and *workspace administrator*.

Developers can create and edit applications. Workspace administrators additionally have access to tools and reports designed to help them manage their workspace. Using the Workspace Administration page, Workspace administrators can create and edit user accounts, monitor workspace activity, review log files, manage session state, view reports, and manage development services. This section describes how to perform these Workspace administrator tasks.

This section contains the following topics:

Understanding Administrator Roles

In a Oracle HTML DB development environment there are two different administrator roles:

- Workspace administrator
- Oracle HTML DB administrator

A Workspace administrator uses the Administration Services page and all the functionality described in this chapter to manage their workspace. In contrast, an Oracle HTML DB administrator manages a complete Oracle HTML DB development environment instance containing multiple workspaces. In order to become a Workspace administrator, an existing administrator must specify the developer as an administrator.

See Also:

- "Creating New User Accounts" on page 11-2 and "Editing Existing User Accounts" on page 11-3 for more information on specifying a developer as an administrator
- "Administering Workspaces" on page 14-1 for more information administering a workspace as an Oracle HTML DB administrator

Managing Users

Workspace administrators can create new user accounts, manage existing user accounts, and change their passwords. User accounts are particularly useful if you are using internal "Cookie User" authentication.

Topics in this section include:

- Creating New User Accounts
- Editing Existing User Accounts
- Changing Your Password

See Also: "About HTML DB Account Credentials" on page 10-14 for more information on implementing internal Cookie User (or HTML DB Account Credentials) authentication

Creating New User Accounts

Workspace administrators can use the Create User page to create new user accounts.

To create a new user account:

1. From the Oracle HTML DB Home page select the **Administration** tab.

2. Under Administration Services, click **Manage Users** and then **Create New User**.

   The Create User page appears.

3. Under User Identification, enter the appropriate information.

4. Under Developer Privileges, specify whether the user is a developer or an administrator.

   Developers having Admin privilege have access to the Administration Services page and all the functionality described in this section. These users can also alter passwords of users within the same workspace.

5. Under User Groups, select an optional user group.

   You can use User Groups to restrict access to various parts of an application. Keep in mind, however, that user groups are not portable over different authentication schemes. They are only useful when using Internal Cookie User authentication.

6. Click **Create User** or **Create and Create Another**.

Editing Existing User Accounts

Workspace administrators can use the Edit User page to edit existing user accounts.

To edit an existing a user account:

1. From the Oracle HTML DB Home page select the **Administration** tab.

2. Under Administration Services, click **Manage Users** and then **Edit Users**.

   The Edit Users page appears.

3. To create a new user, click **Create**.

4. To find and edit an existing user, enter a search condition and click **Go**. Once the user appears, click the edit icon to edit the account.

   The Edit User page appears.

5. Follow the on-screen instructions.

Changing Your Password

Workspace administrators can use the Change Password page to change their password.

To change your password:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Under Administration Services, click **Manage Users** and then **Change My Password**.

   The Change Password page appears.
3. Type a new password in the fields provided and click **Apply Changes**.

Monitoring Users

As a Workspace administrator, you can monitor workspace utilization and user activity by accessing a number of charts and reports on the Monitor page.

To view workspace utilization and user activity reports:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Under Administration Services, click **Monitor**.

   The Monitor page appears. It is divided into three sections:

   - Chart Activity
   - Activity Reports
   - Calendar of Activity
3. Select a chart or report to review.

Administering Session State and User Preferences

A session is a logical construct that establishes persistence (or stateful behavior) across page views. Each session is assigned a unique ID which the HTML DB engine uses to store and retrieve an application's working set of data (or session state) before and after each page view. Sessions persist in the database until purged by an administrator.

Workspace administrators can purge session state or user preferences within their workspace on the Session State Management page.

Topics in this section include:

- Managing Session State and User Preferences for the Current Session
- Managing Recent Sessions

See Also:

- "Understanding Session State Management" on page 6-12
- "Managing User Preferences" on page 12-18
- "Managing Session State" on page 15-3

Managing Session State and User Preferences for the Current Session

Use the Current Session page to manage session state and user preferences for the current session.

To access the Current Session page:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Under Administration Services, click **Session State**.

   The Session State Management page appears.
3. Click **Report, with an option to purge, your current session**.

   The Current Session Page appears
4. Under Session State you can:
   - Reset the session state for the current session by clicking **Purge Session State**
   - View information about the current session by clicking **View Session State**
5. Under User Preferences, you can:
   - View preferences for the current user, by clicking **View Preferences**
   - Reset user preferences for the current user by clicking **Reset Preferences**

See Also: "Viewing Session State" on page 6-13

Managing Recent Sessions

Workspace administrators can determine whether to purge existing sessions by either:

- Purging sessions by age
- First reviewing session details and then optionally purging selected sessions

To purge existing session by age:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Under Administration Services, click **Session State**.

   The Session State Management page appears.
3. Select **Purge existing sessions by age**.
4. From the Sessions older than list, select a time increment and click either:
   - **Purge Sessions**
   - **Report Session**

To first review session details and then purge the session:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Under Administration Services, click **Session State**.

   The Session State Management page appears.
3. Select **Report recent sessions with drilldown to session details**.
4. Select a session ID.
5. When Session Information appears you click either:
   - **Remove State**
   - **Remove Session**

Viewing Workspace Reports

Workspace administrators can view a variety of application and administrative reports on the Administrative Reports page.

To view workspace administrative reports:

1. From the Application Builder Home page select the **Administration** tab.
2. Under Administration Services, click **Reports**.

   The Administrative Reports page appears.
3. Select a report to review.

Monitoring Developer Activity

Workspace administrators can view a variety of application and administrative reports on the Administrative Reports page.

To view workspace administrative reports:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Under Administration Services, click **Monitor**.

   The Monitor page appears.
3. Select a report to review.

Managing Log Files

Workspace administrators can manage the following log files:

- Developer activity logs
- External click counting log

To view developer activity:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Under Administration Services, click **Logs**.

   The Log Files page appears.
3. Click **Monitor Developer Activity**.
4. Specify a time frame and the appropriate number of rows and click **Go**.
5. To view additional details, select a developer.

To purge Developer activity logs:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Under Administration Services, click **Logs**.
3. Click **Purge Dev. Log**.

To purge the External click counting log:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Under Administration Services, click **Logs**.
3. Click **Purge Click. Log**.

Managing Development Services

Workspace administrators can use the Provisioning Services section of the Administration Services page to:

- View information and reports describing the current workspace
- Submit a request to the Oracle HTML DB administrator for a new database schema, additional storage, or to terminate workspace service

Topics in this section include:

Viewing Current Workspace Status

Workspace administrators can view current workspace status on the Manage Development Services page.

To view current workspace status:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Click **Provisioning Services** and then **Manage Service**.

   The Manage Development Services page appears.
3. Select Report Utilization.
4. Follow the on-screen instructions.

Requesting a Database Schema

To submit a request to the Oracle HTML DB administrator for a new database schema:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Click **Provisioning Services** and then **Manage Service**.

   The Manage Development Services page appears.
3. Select **Request Schema**.
4. Follow the on-screen instructions.

Requesting Additional Storage

To submit a request to the Oracle HTML DB administrator for additional storage space for your workspace:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Click **Provisioning Services** and then **Manage Service**.

   The Manage Development Services page appears.
3. Select Request Storage.
4. Follow the on-screen instructions.

Requesting Service Termination

To submit a request to the Oracle HTML DB administrator to terminate workspace service:

1. From the Oracle HTML DB Home page select **Administration** tab.
2. Click **Provisioning Services** and then **Manage Service**.

   The Manage Development Services page appears.
3. Select **Terminate Service**.
4. Follow the on-screen instructions.

12

Advanced Programming Techniques

This section provides information about advanced programming techniques including establishing database links, using collections, running background SQL, utilizing Web Services and managing user preferences.

This section contains the following topics:

- Accessing Data with Database Links
- Using Collections
- Running Background PL/SQL
- Implementing Web Services
- Managing User Preferences

See Also: "Oracle HTML DB APIs" on page 13-1

Accessing Data with Database Links

Since Oracle HTML DB runs on top of an Oracle database, you have access to all distributed database capabilities. Typically, you perform distributed database operations using database links.

To use database links, you must create a standard database link using the following standard Oracle syntax:

```
CREATE DATABASE LINK linkname
CONNECT TO username IDENTIFIED BY password
USING 'tns_connect_string';
```

The `tns_connect_string` entry on your local server should correspond to the information in your SERVERS tnsnames.ora file. It is a good idea to name your database link the global name of the remote database.

See Also: *Oracle Database Administrator's Guide*

Using Collections

Collections enable you to temporarily capture one or more non-scalar values. You can use collections to store rows and columns currently in session state so they can be accessed, manipulated, or processed during a user's specific session. Think of a collection as a bucket in which you can temporarily store and name rows of information.

Examples of when you might use collections include:

- When you are creating a data-entry wizard in which multiple rows of information first need to be collected within a logical transaction. You can use collections to temporarily store the contents of the multiple rows of information, prior to performing the final wizard step when both the physical and logical transactions are completed.
- When your application includes an update page on which a user updates multiple detail rows on one page. They can make many updates, apply these updates to a collection, then call a final process to apply the changes to the database.
- When you are building a wizard where you are collecting an arbitrary number of attributes. At the end of the wizard the user then performs a task that takes the information temporarily stored in the collection and applies it to the database.

Using the HTMLDB_COLLECTION API

You implement a collection using the PL/SQL API `HTMLDB_COLLECTION`. Using this API you can insert, update, and delete collection information.

Topics in this section include:

- Updating Collection Members
- Deleting a Collection Member
- Determining Collection Status
- Merging Collections
- Managing Collections
- Clearing Collection Session State

About Collection Naming

When you create a new collection, you must give it a name that cannot exceed 255 characters. Note that collection names are not case sensitive and will be converted to upper case.

Once named, you can access the values in a collection by running a SQL query against the view `HTMLDB_COLLECTION`.

Creating a Collection

Every collection contains a named list of data elements (or members) which can have up to 50 attributes (or columns). Use the following methods to create a collection:

- `CREATE_COLLECTION`
- `CREATE_OR_TRUNCATE_COLLECTION`
- `CREATE_COLLECTION_FROM_QUERY`

`CREATE_COLLECTION` raises an exception if the named collection already exists. For example:

```
HTMLDB_COLLECTION.CREATE_COLLECTION(
    p_collection_name => collection name );
```

`CREATE_OR_TRUNCATE_COLLECTION` creates a new collection if the named collection does not exist. If the named collection already exists, this method truncates it. Truncating a collection empties it, but leaves it in place. For example:

```
HTMLDB_COLLECTION.CREATE_OR_TRUNCATE_COLLECTION(
    p_collection_name => collection name );
```

`CREATE_COLLECTION_FROM_QUERY` creates a collection then populates it with the results of a specified query. For example:

```
HTMLDB_COLLECTION.CREATE_COLLECTION_FROM_QUERY(
    p_collection_name => collection name,
    p_query           => your query );
```

Truncating a Collection

Truncating a collection removes all members from the specified collection, but leaves the named collection in place. For example:

```
HTMLDB_COLLECTION.TRUNCATE_COLLECTION(
    p_collection_name => collection name );
```

Deleting a Collection

Deleting a collection deletes the collection and all of its members. Be aware that if you do not delete a collection, it will eventually be deleted when the session is purged. For example:

```
HTMLDB_COLLECTION.DELETE_COLLECTION (
    p_collection_name => collection name );
```

Deleting All Collections for the Current Application Use the method `DELETE_ALL_COLLECTIONS` to delete all collections defined in the current application. For example:

```
HTMLDB_COLLECTION.DELETE_ALL_COLLECTIONS;
```

Deleting All Collections in the Current Session Use the method `DELETE_ALL_COLLECTIONS_SESSION` to delete all collections defined in the current session. For example:

```
HTMLDB_COLLECTION.DELETE_ALL_COLLECTIONS_SESSION;
```

Adding Members to a Collection

When data elements (or members) are added to a collection, they are assigned a unique sequence ID. As you add members to a collection, the sequence ID will change in increments of 1 with the newest members having the largest ID.

You add new member to a collection using the function ADD_MEMBER. Calling this method returns the sequence ID of the newly added member. The following example demonstrates how to use the procedure ADD_MEMBER.

```
HTMLDB_COLLECTION.ADD_MEMBER(
    p_collection_name => collection name,
    p_c001          => [member attribute 1],
    p_c002          => [member attribute 2],
    p_c003          => [member attribute 3],
    p_c004          => [member attribute 4],
    p_c005          => [member attribute 5],
    p_c006          => [member attribute 6],
    p_c007          => [member attribute 7],
  ...
    p_c050          => [member attribute 50]);
```

The next example demonstrates how to use the function ADD_MEMBER. This function returns the sequence number assigned to the newly created member.

```
l_id := HTMLDB_COLLECTION.ADD_MEMBER(
    p_collection_name => collection name,
    p_c001          => [member attribute 1],
    p_c002          => [member attribute 2],
    p_c003          => [member attribute 3],
    p_c004          => [member attribute 4],
    p_c005          => [member attribute 5],
    p_c006          => [member attribute 6],
    p_c007          => [member attribute 7],
  ...
    p_c050          => [member attribute 50]);
```

You can also add new members (or an array of members) to a collection using the method ADD_MEMBERS. This method raises an exception if the specified collection does not exist with the specified name of the current user and in the same session. Also any attribute exceeding 4,000 characters will be truncated to 4,000 characters. The number of members added is based on the number of elements in the first array. For example:

```
HTMLDB_COLLECTION.ADD_MEMBERS(
    p_collection_name => collection name,
    p_c001          => member attribute array 1,
    p_c002          => member attribute array 2,
    p_c003          => member attribute array 3,
    p_c004          => member attribute array 4,
```

```
    p_c005          => member attribute array 5,
    p_c006          => member attribute array 6,
    p_c007          => member attribute array 7,
    ...
    p_c050          => member attribute array 50);
```

Updating Collection Members

You can update collection members by calling UPDATE_MEMBER and referencing the desired collection member by its sequence ID. This procedure replaces an entire collection member, not individual member attributes. This procedure raises an exception if the named collection does not exist. For example:

```
HTMLDB_COLLECTION.UPDATE_MEMBER (
    p_collection_name => collection name,
    p_seq             => member sequence number,
    p_c001            => member attribute 1,
    p_c002            => member attribute 2,
    p_c003            => member attribute 3,
    p_c004            => member attribute 4,
    p_c005            => member attribute 5,
    p_c006            => member attribute 6,
    p_c007            => member attribute 7,
    ...
    p_c050            => member attribute 50);
```

If you wish to update a single attribute of a collection member, use UPDATE_MEMBER_ATTRIBUTE. Calling this procedure raises an exception if the named collection does not exist. For example:

```
HTMLDB_COLLECTION.UPDATE_MEMBER_ATTRIBUTE(
    p_collection_name => collection name,
    p_seq             => member sequence number,
    p_attr_number     => number of attribute to be updated,
    p_attr_value      => new attribute value);
```

Deleting a Collection Member

You can delete a collection member by calling DELETE_MEMBER and referencing the desired collection member by its sequence ID. For example:

```
HTMLDB_COLLECTION.DELETE_MEMBER(
    p_collection_name => collection name,
    p_seq             => member sequence number);
```

Be aware that this procedure leaves a gap in the sequence IDs in the specified collection. Also, calling this procedure results in an error if the named collection does not exist.

You can also delete all members from a collection by when an attribute matches a specific value. For example:

```
HTMLDB_COLLECTION.DELETE_MEMBERS(
    p_collection_name => collection name,
    p_attr_number     => number of attribute to be updated,
    p_attr_value      => new attribute value);
```

Be aware that this procedure also leaves a gap in the sequence IDs in the specified collection. Also, this procedure raises an exception if:

- The named collection does not exist
- The specified attribute number is outside the range of 1 to 50, or is in invalid

If the supplied attribute value is null, then all members of the named collection will deleted.

Determining Collection Status

Each collection contains a flag to determine if the contents of the collection has changed. This flag is set when you first create a collection by calling `CREATE_COLLECTION` or `CREATE_OR_TRUNCATE_COLLECTION`. You can reset this flag manually by calling `RESET_COLLECTION_CHANGED`. For example:

```
HTMLDB_COLLECTION.RESET_COLLECTION_CHANGED (
    p_collection_name => collection name)
```

Once this flag has been reset, you can determine if a collection has changed by calling `COLLECTION_HAS_CHANGED`. For example:

```
l_changed := HTMLDB_COLLECTION.COLLECTION_HAS_CHANGED(
  p_collection_name => collection_name);
```

When you add a new member to a collection, an MD5 message is automatically computed against all 50 attributes of the member. You can access this value from the MD5_ORIGINAL column of the view `HTMLDB_COLLECTION` using the function `GET_MEMBER_MD5`. For example:

```
HTMLDB_COLLECTION.GET_MEMBER_MD5 (
    p_collection_name => collection name,
    p_seq             => member sequence number );
```

```
    RETURN VARCHAR2;
```

Merging Collections

You can merge members of collection with values passed in a set of arrays. By using the argument `p_init_query`, you can create a collection from the supplied query. Be aware, however, that if the collection exists, the following occurs:

- Rows in the collection (not in the arrays) will be deleted
- Rows in the collection and in the arrays will be updated
- Rows in the array and not in the collection will be inserted

Any attribute value exceeding 4,000 characters will be truncated to 4,000 characters. Table 12–1 describes the available arguments you can use when merging collections.

Table 12–1 Available Arguments for Merging Collections

Argument	Description
`p_c001`	Array of first attribute values to be merged. Maximum length can be 4,000 characters. If the maximum length is greater, it will be truncated to 4,000 characters. The count of elements in the P_C001 PL/SQL table is used as the total number of items across all PL/SQL tables. For example, if P_C001.count = 2 and P_C002.count = 10, only 2 members will be merged. Be aware that if P_C001 is null, an application error will be raised.
`p_c0xx`	Attribute of `XX` attributes values to be merged. Maximum length can be 4,000 characters. If the maximum length is greater, it will be truncated to 4,000 characters.
`p_collection_name`	Name of the collection. **See Also:** "About Collection Naming" on page 12-3
`p_null_index`	Use this argument to identify rows the merge function should ignore. This argument identifies an row as null. Null rows are automatically removed from the collection. Use `p_null_index` in conjunction with.
`p_null_value`	Use this argument in conjunction with the `p_null_index`. Identifies the null value. If used this value cannot be null. A typical value for this argument is 0.
`p_init_query`	Use the query defined by this argument to create a collection if the collection does not exist.

Managing Collections

You can use the following utilities to manage collections.

Obtaining a Member Count Use `COLLECTION_MEMBER_COUNT` to return the total count of all members in a collection. Be aware that this count does not imply the highest sequence in the collection. For example:

```
l_count := HTMLDB_COLLECTION.COLLECTION_MEMBER_COUNT (
  p_collection_name => collection name );
```

Resequencing a Collection Use `RESEQUENCE_COLLECTION` to resequence a collection to remove any gaps in sequence IDs while maintaining the same element order. For example:

```
HTMLDB_COLLECTION.RESEQUENCE_COLLECTION (
   p_collection_name => collection name )
```

Verifying Whether a Collection Exists Use `COLLECTION_EXISTS` to determine if a collection exists. For example:

```
l_exists := HTMLDB_COLLECTION.COLLECTION_EXISTS  (
  p_collection_name => collection name );
```

Adjusting Member Sequence ID You can adjust the sequence ID of a specific member within a collection by moving the ID up or down. When you adjust a sequence ID, the specified ID is exchanged with another one. For example, if you were to move the ID 2 up, 2 would become 3 and 3 would become 2.

Use `MOVE_MEMBER_UP` to adjust a member sequence ID up by one. Alternately, use `MOVE_MEMBER_DOWN` to adjust a member sequence ID down by one. For example:

```
HTMLDB_COLLECTION.MOVE_MEMBER_DOWN(
    p_collection_name => collection name,
    p_seq             => member sequence number);
```

Be aware that while using either of these methods an application error displays:

- If the named collection does not exist for the current user in the current session
- If the member specified by sequence ID `p_seq` does not exist

However, an application error will not be returned if the specified member already has the highest or lowest sequence ID in the collection (depending on whether you are calling `MOVE_MEMBER_UP` or `MOVE_MEMBER_DOWN`).

Sorting Collection Members Use `SORT_MEMBERS` to reorder members of a collection by the column number. This method not only sorts the collection by a particular column number, but it also reassigns the sequence IDs for each member to remove gaps. For example:

```
HTMLDB_COLLECTION.SORT_MEMBERS(
    p_collection_name       => collection name,
    p_sort_on_column_number => column number to sort by);
```

Clearing Collection Session State

By clearing the session state of a collection, you remove the collection members. A shopping cart is a good example of when you might need to clear collection session state. When a user requests to empty his or her cart and start again, you would need to clear the session state for a collection. You can remove session state of a collection by calling the `CREATE_OR_TRUNCATE_COLLECTION` method or by using `f?p` syntax.

Calling `CREATE_OR_TRUNCATE_COLLECTION` deletes the existing collection and then recreates it. For example:

```
HTMLDB_COLLECTION.CREATE_OR_TRUNCATE_COLLECTION(
    p_collection_name       => collection name,
```

You can also use the sixth `f?p` syntax argument to clear session state. For example:

```
f?p=App:Page:Session::NO:1,2,3,collection name
```

See Also: "Understanding URL Syntax" on page 6-19

Running Background PL/SQL

You can use the `HTMLDB_PLSQL_JOB` package to run PL/SQL code in the background of your application. This is an effective approach for managing long running operations that do not need to complete in order for a user to continue working with your application.

Topics in this section include:

- Understanding the HTMLDB_PLSQL_JOB Package
- About System Status Updates
- Using a Process to Implement Background PL/SQL

Understanding the HTMLDB_PLSQL_JOB Package

HTMLDB_PLSQL_JOB is a wrapper package around DBMS_JOB functionality offered in the Oracle database. Be aware that the HTMLDB_PLSQL_JOB package only exposes that functionality which is necessary to run PL/SQL in the background. The following is a description of the HTMLDB_PLSQL_JOB package.

```
SQL> DESC HTMLDB_PLSQL_JOB
FUNCTION JOBS_ARE_ENABLED RETURNS BOOLEAN
PROCEDURE PURGE_PROCESS
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 P_JOB                          NUMBER                  IN
FUNCTION SUBMIT_PROCESS RETURNS NUMBER
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 P_SQL                          VARCHAR2                IN
 P_WHEN                         VARCHAR2                IN     DEFAULT
 P_STATUS                       VARCHAR2                IN     DEFAULT
FUNCTION TIME_ELAPSED RETURNS NUMBER
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 P_JOB                          NUMBER                  IN
PROCEDURE UPDATE_JOB_STATUS
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 P_JOB                          NUMBER                  IN
 P_STATUS                       VARCHAR2                IN
 P_DESC
```

Table 12–1 describes the functions available in the HTMLDB_PLSQL_JOB package.

Table 12–2 HTMLDB_PLSQL_JOB Package Available Functions

Function	Description
SUBMIT_PROCESS	Use this procedure to submit background PL/SQL. This procedure returns a unique job number. Since you can use this job number as a reference point for other procedures and functions in this package, it may be useful to store it in your own schema.
UPDATE_JOB_STATUS	Call this procedure to update the status of the currently running job. This procedure is most effective when called from the submitted PL/SQL.

Table 12–2 HTMLDB_PLSQL_JOB Package Available Functions

Function	Description
TIME_ELAPSED	Use this function to determine how much time has elapsed since the job was submitted.
JOBS_ARE_ENABLED	Call this function to determine whether or not that database is currently in a mode which supports submitting jobs to the HTMLDB_PLSQL_JOB package.
PURGE_PROCESS	Call this procedure to clean up submitted jobs. Submitted jobs stay in the HTMLDB_PLSQL_JOBS view until either Oracle HTML DB cleans out those records, or you call PURGE_PROCESS to manually remove them.

You can view all jobs submitted to the HTMLDB_PLSQL_JOB package using the HTMLDB_PLSQL_JOBS view. The following is the description of HTMLDB_PLSQL_JOBS view.

```
SQL> DESCRIBE HTMLDB_PLSQL_JOBS
 Name                              Null?    Type
 --------------------------------- -------- ----------------------------
 ID                                         NUMBER
 JOB                                        NUMBER
 FLOW_ID                                    NUMBER
 OWNER                                      VARCHAR2(30)
 ENDUSER                                    VARCHAR2(30)
 CREATED                                    DATE
 MODIFIED                                   DATE
 STATUS                                     VARCHAR2(100)
 SYSTEM_STATUS                              VARCHAR2(4000)
 SYSTEM_MODIFIED                            DATE
 SECURITY_GROUP_ID                          NUMBER
```

Table 12–3 describes the columns available in HTMLDB_PLSQL_JOBS view.

Table 12–3 HTMLDB_PLSQL_JOBS View Columns

Name	Description
ID	An unique identifier for each row.
JOB	The job number assigned to each submitted PL/SQL job. The HTMLDB_PLSQL_JOB.SUBMIT_PROCESS function returns this value. This is also the value you pass into other procedures and functions in the HTMLDB_PLSQL_JOB package.

Table 12–3 HTMLDB_PLSQL_JOBS View Columns

Name	Description
FLOW_ID	The application from which this job was submitted.
OWNER	The database schema that owns the application. This identifies what schema will parse this code when DBMS_JOB runs it.
ENDUSER	The end user (that is, who logged into the application) that caused this process to be submitted.
CREATED	The date when the job was submitted.
MODIFIED	The date when the status was modified.
STATUS	The user defined status for this job. Calling HTMLDB_PLSQL_JOB.UPDATE_JOB_STATUS updates this column.
SYSTEM_STATUS	The system defined status for this job.
SYSTEM_MODIFIED	The date when the system status was modified.
SECURITY_GROUP_ID	The unique ID assigned to your workspace. Developers can only see jobs submitted from their own workspace.

About System Status Updates

Submitted jobs can contain any of the following system status settings:

- **SUBMITTED.** Indicates the job has been submitted, but has not yet started. DBMS_JOB does not guarantee immediate starting of jobs.
- **IN PROGRESS.** Indicates that DBMS_JOB has started the process.
- **COMPLETED.** Indicates the job has finished.
- **BROKEN (sqlcode) sqlerrm.** Indicates there was a problem in your job that resulted in an exception. The SQL code and SQL Error Message for the exception should be included in the system status. Review this information to determine what went wrong.

Using a Process to Implement Background PL/SQL

The simplest way to implement the HTMLDB_PLSQL_JOB package is to create a page process that specifies the process type PLSQL DBMS JOB. By selecting this process type, Application Builder will submit the PL/SQL code you specify as a job. Since you are not calling the function directly, you can use the built-in substitution item APP_JOB to determine the job number of any jobs you submit.

The following example runs a PL/SQL job in the background for testing and explanation.

```
001  BEGIN
002    FOR i IN 1 .. 100 LOOP
003      INSERT INTO emp(a,b) VALUES (:APP_JOB,i);
004      IF MOD(i,10) = 0 THEN
005        HTMLDB_PLSQL_JOB.UPDATE_JOB_STATUS(
006          P_JOB     => :APP_JOB,
007          P_STATUS  => i || 'rows inserted');
008      END IF;
009      HTMLDB_UTIL.PAUSE(2);
010    END LOOP;
011  END;
```

In this example, note that:

- Lines 002 to 010 run a loop that inserts 100 records into the `emp` table.
- `APP_JOB` is referenced as a bind variable inside the VALUES clause of the INSERT, and specified as the `P_JOB` parameter value in the call to `UPDATE_JOB_STATUS`.
- `APP_JOB` represents the job number which will be assigned to this process as it is submitted to `HTMLDB_PLSQL_JOB`. By specifying this reserved item inside your process code, it will be replaced for you at execution time with the actual job number.
- Notice this example calls to `UPDATE_JOB_STATUS` every ten records, INSIDE the block of code. Normally, Oracle transaction rules dictate updates made inside code blocks will not be seen until the entire transaction is committed. The `HTMLDB_PLSQL_JOB.UPDATE_JOB_STATUS` procedure, however, has been implemented in such a way that the update will happen regardless of whether or not the job succeeds or fails. This last point is important for two reasons:

  1. Even if your status reads "100 rows inserted," it does not mean the entire operation was successful. If an exception occurred at the time the block of code tried to commit, the `user_status` column of `HTMLDB_PLSQL_JOBS` would not be affected since status updates are committed separately.
  2. These updates are performed autonomously. You can view the job status before the job has completed. This gives you the ability to display status text about ongoing operations in the background as they are happening.

Implementing Web Services

Web services in Oracle HTML DB are based on SOAP (the Simple Object Access Protocol). SOAP is a World Wide Web Consortium (W3C) standard protocol for sending and receiving requests and responses across the Internet. SOAP messages can be sent back and forth between a service provider and a service user in SOAP envelopes. SOAP envelopes contain a request for some action and the result of that action and are formatted in XML.

Because SOAP is based on XML and uses simple transport protocols such as HTTP, SOAP messages are not blocked by firewalls and is very easy to use. A SOAP message consists of the following:

- An envelope that contains the message, defines how to process the message, who should process the message, and whether processing is optional or mandatory.
- Encoding rules that describe the data types for the application. These rules define a serialization mechanism that converts the application data types to XML and XML to data types.
- Remote procedure call definitions.

Note that the SOAP 1.1 specification is a W3C note. (The W3C XML Protocol Working Group has been formed to create a standard that will supersede SOAP.)

> **See Also:** For more information on Simple Object Access Protocol (SOAP) 1.1 see:
>
> `http://www.w3.org/TR/SOAP/`

Creating a Web Service

To create a Web service in Oracle HTML DB, you must provide:

- The URL used to post the SOAP request over HTTP
- An URI (Uniform Resource Identifier) identifying the SOAP HTTP request
- A Proxy address
- A SOAP envelope

To create a new Web service:

1. Click the **Build** icon.

2. From the Available Applications list, select an application and click **Go**.
3. Select the **WebServices** tab.

   The Web Services pages appears. Existing services display in the Web Services Repository. You can test an existing service by clicking the service name.
4. To create a new Web service, click **Create**.

   The Create/Edit Web Service page appears. Required attributes are identified with a red asterisk (*).
5. In Web Service Identification, enter a name for this Web service. This name only appears within the context of Application Builder.
6. Under Service Description, specify the following:
   - In URL, specify the URL used to post the SOAP request over HTTP. This URL corresponds to the soap:address location of a service port in the WSDL (Web Services Description Language). For example:

     ```
     http://www.alethea.net/webservices/LocalTime.asmx
     ```

   - In Action, indicate the intent of the SOAP HTTP request. This value is a URI (Uniform Resource Identifier) identifying the intent. SOAP places no restrictions on the format or specificity of the URI or whether or not it is resolvable. An HTTP client must use this header field when issuing a SOAP HTTP Request.
   - In Proxy, enter a proxy using the following syntax:

     ```
     http://host:port/
     ```

     This proxy overrides the system defined HTTP proxy for your request and may include an optional TCP/IP port number at which the proxy server listens. For example,

     ```
     www-proxy.myworkspace.com
     ```

7. In SOAP Envelope, specify the SOAP envelope to be used for the SOAP request to the Web service. This envelope can contain item substitutions using the syntax `#ITEM_NAME#`. For example:

   ```
   <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/1999/XMLSchema"
     SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   ```

```
  <SOAP-ENV:Body><LocalTimeByZipCode
xmlns="http://www.alethea.net/webservices/"><ZipCode
xsi:type="xsd:string">#Fxxxx_Pyyy_ZIP_
CODE#</ZipCode></LocalTimeByZipCode></SOAP-ENV:Envelope>
```

8. Under Style Sheet Rendering, enter an enter a valid XSL style sheet. This style sheet:
   - Renders the result of the Web Service in a page region.
   - Is used to apply an XML transformation against the result of the SOAP request.
   - Render the output in a Web Service page region.
9. Click **Create**.

Invoking a Web Service as a Process

You can also implement a Web Service as a process on the page. Running the process submits the request to the service provider. You can then display the request results in Web Service region and use a style sheet to render the output.

To invoke a Web Service as a process:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Under Page Processes, click **Create**.

   The Create Page Computation Wizard appears.
3. Specify a process name, sequence, and processing point.
4. From Type, select **Web Service**.
5. Identify the Web Service.
6. Follow the on-screen instructions

To create a Web Service region:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Under Regions, click **Create**.

   The Create Region Wizard appears.
3. Select the region type **Other** and then select **Web Service Result**.

4. Follow the on-screen instructions.

Managing User Preferences

You can use preferences to set the session state for a specific user. Once set, these preferences can only be removed by an Oracle HTML DB administrator. You can set user preferences by creating a page process, by the calculation of a preference Item Source Value, or programatically using the PL/SQL API.

Topics in this section include:

- Viewing User Preferences
- Setting User Preferences
- Resetting User Preferences Manually
- Resetting Preferences Using a Page Process

Viewing User Preferences

You view user preferences for a specific user on the Session State Management page.

To view user preferences for a specific user:

1. From the Oracle HTML DB Home page select the **Administration** tab.
2. Under Administration Services, click **Manage Users** and then **Session State**.

   The Session State Management page appears.
3. Click **Report preferences for users**.
4. Type a username in the field provided and click **Go**.

> **See Also:** "Administering Session State and User Preferences" on page 11-4 for more information on using the Session State Management page

Setting User Preferences

You can set user preferences within your application through the creation of a page process, by creating a preference item, or programatically.

Topics in this section include:

- Setting User Preferences Using a Page Process

- Setting the Source of an Item Based on a User Preference
- Setting User Preferences Programatically

Setting User Preferences Using a Page Process

To set user preference values by creating a page process:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Under Page Processes, click **Create**.

   The Create Page Computation Wizard appears.
3. Specify a process name, sequence, and processing point.
4. From Type, select one of the following:
   - **Set Preference to value of item**
   - **Set Preference to value of item if item is not NULL**
5. Specify the preference value in the field provided using the format:

   ```
   PreferenceName:Item
   ```
6. Click **Page Items** to see a list of available items.
7. Follow the on-screen instructions

Setting the Source of an Item Based on a User Preference

You can set the source of an item based on a user preference by defining the item source type as Preference.

To define the source of item based on a user preference:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Under Item, click **Create**.

   The Create Page Computation Wizard appears.
3. Specify the Item Name and Display Position Attributes and click **Next**.
4. Specify the Item Attributes click **Next**.
5. From the Item Source list, select **Preferences**.
6. In Item Source Value, enter the name of the preference.

7. Follow the on-screen instructions

Setting User Preferences Programatically

To set or reference user preferences programatically, you must use a PL/SQL API. User level caching is available programmatically. You can use the `set_preferences` function to set a user level preference called `NAMED_PREFERENCE`. For example:

```
HTMLDB_UTIL.SET_PREFERENCE(
 p_preference=>'NAMED_PREFERENCE',
 p_value     =>v('ITEM_NAME'));
```

You can reference the value of a user preference using the function `GET_PREFERENCES`. For example:

```
NVL(HTMLDB_UTIL.GET_PREFERENCE('NAMED_PREFERENCE'),15)
```

In the previous example, the preference would default to the value 15 if the preference contained no value.

> **See Also:** "GET_PREFERENCE Function" on page 13-6 and "SET_PREFERENCE Procedure" on page 13-9

Resetting User Preferences Manually

You can manually purge user preferences for a specific user.

To manually purge preferences for a specific user:

1. From the Oracle HTML DB Home page select **Administration** tab.
2. Under Administration Services, click **Manage Users** and then **Session State**.

   The Session State Management page appears.
3. Click **Purge preferences for a selected user**.
4. Specify a user and follow the on-screen instructions.

> **See Also:** "Administering Session State and User Preferences" on page 11-4 for more information on using the Session State Management page

Resetting Preferences Using a Page Process

You can reset user preferences by creating a page process and selecting the process type Reset Preferences.

To reset user preferences using a page process:

1. Navigate to the appropriate Page Definition. (See "Viewing a Page Definition" on page 7-5.)
2. Under Page Processes, click **Create**.

   The Create Page Computation Wizard appears.
3. Specify a process name, sequence, and processing point.
4. From Type, select **Reset Preferences**.
5. Follow the on-screen instructions

13

Oracle HTML DB APIs

This section describes the APIs available in Oracle HTML DB.

This section contains the following topics:

HTMLDB_UTIL

The `HTMLDB_UTIL` package provides utilities you can use when programming in the Oracle HTML DB environment. You can use `HTMLDB_UTIL` to get and set session state, get files, check authorizations for users, reset different states for users, and also to get and set preferences for users.

Topics in this section include:

CLEAR_APP_CACHE Procedure

This procedure removes session state for a given application for the current session.

Syntax

```
HTMLDB_UTIL.CLEAR_APP_CACHE (
    p_app_id    IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 13–1 describes the parameters available in the `CLEAR_APP_CACHE` procedure.

Table 13–1 CLEAR_APP_CACHE Parameters

Parameter	Description
`p_app_id`	The ID of the application for which session state will be cleared for current session.

Example

```
BEGIN
        HTMLDB_UTIL.CLEAR_APP_CACHE('100');
END;
```

CLEAR_USER_CACHE Procedure

This procedure removes session state and application system preferences for the current user's session. Run this procedure if you reuse session IDs and want to run applications without the benefit of existing session state.

Syntax

```
HTMLDB_UTIL.CLEAR_USER_CACHE;
```

Example

```
BEGIN
       HTMLDB_UTIL.CLEAR_USER_CACHE;
END;
```

COUNT_CLICK Procedure

This procedure counts clicks from an Oracle HTML DB application to an external site. You can also use the shorthand version procedure `Z` in place of `HTMLDB_UTIL.COUNT_CLICK`.

Syntax

```
HTMLDB_UTIL.COUNT_CLICK (
    p_url          IN    VARCHAR2,
    p_cat          IN    VARCHAR2,
    p_id           IN    VARCHAR2    DEFAULT NULL,
    p_user         IN    VARCHAR2    DEFAULT NULL,
    p_company      IN    VARCHAR2    DEFAULT NULL);
```

Parameters

Table 13–2 describes the parameters available in the `COUNT_CLICK` procedure.

Table 13–2 COUNT_CLICK Parameters

Parameter	Description
`p_url`	The URL to redirect to.
`p_cat`	A category to classify the click.
`p_id`	Secondary ID to associate with the click (optional).

Table 13–2 COUNT_CLICK Parameters

Parameter	Description
p_user	The application user ID (optional).
p_company	The workspace associated with the application (optional).

Example

```
BEGIN
htp.p('<a
href="HTMLDB_UTIL.COUNT_CLICK?p_url=http://yahoo.com&p_cat=yahoo">Click
here</a>');
END;
```

GET_FILE Procedure

This procedure downloads files from the Oracle HTML DB file repository.

Syntax

```
HTMLDB_UTIL.GET_FILE (
    p_file_id    IN   VARCHAR2,
    p_mime_type  IN   VARCHAR2 DEFAULT NULL,
    p_inline     IN   VARCHAR2 DEFAULT 'NO');
```

Parameters

Table 13–3 describes the parameters available in GET_FILE procedure.

Table 13–3 GET_FILE Parameters

Parameter	Description
p_file_id	ID in HTMLDB_APPLICATION_FILES of the file to be downloaded.
p_mime_type	Mime type of the file to download.
p_inline	Valid values include YES and NO. YES to display inline in a browser. NO to download as attachment.

Example

```
BEGIN
        HTMLDB_UTIL.GET_FILE(
              p_file_id   => '8675309',
              p_mime_type => 'text/xml',
              p_inline    => 'YES');
END;
```

GET_NUMERIC_SESSION_STATE Function

This function returns a numeric value for a numeric item. You can use this function in Oracle HTML DB applications wherever you can use PL/SQL or SQL. You can also use the shorthand, function NV, in place of HTMLDB_UTIL.GET_NUMERIC_SESSION_STATE.

Syntax

```
HTMLDB_UTIL.GET_NUMERIC_SESSION_STATE (
    p_item     IN VARCHAR2)
    RETURN NUMBER;
```

Parameters

Table 13–4 describes the parameters available in the GET_NUMERIC_SESSION_STATE function.

Table 13–4 GET_SESSION_STATE Parameters

Parameter	Description
p_item	Case insensitive name of the item for which you wish to have the session state fetched.

Example

```
DECLARE
      l_item_value    Number;
BEGIN
      l_item_value := HTMLDB_UTIL.GET_NUMERIC_SESSION_STATE('my_item');
END;
```

GET_PREFERENCE Function

This function retrieves the value of a previously saved preference for a given user.

Syntax

```
HTMLDB_UTIL.GET_PREFERENCE (
    p_preference  IN    VARCHAR2 DEFAULT NULL,
    p_user        IN    VARCHAR2 DEFAULT V('USER'))
    RETURN VARCHAR2;
```

Parameters

Table 13–5 describes the parameters available in the GET_PREFERENCE function.

Table 13–5 GET_PREFERENCE Parameters

Parameter	Description
p_preference	Name of the preference to retrieve the value.
p_value	Value of the preference.
p_user	User for whom the preference is being retrieved.

Example

```
DECLARE
      l_default_view    VARCHAR2(255);
BEGIN
      l_default_view := HTMLDB_UTIL.GET_PREFERENCE(
                  p_preference => 'default_view',
                  p_user       => :APP_USER);
END;
```

GET_SESSION_STATE Function

This function returns the value for an item. You can use this function in your Oracle HTML DB applications wherever you can use PL/SQL or SQL. You can also use the shorthand, function V, in place of HTMLDB_UTIL.GET_SESSION_STATE.

Syntax

```
HTMLDB_UTIL.GET_SESSION_STATE (
    p_item    IN   VARCHAR2)
    RETURN VARCHAR2;
```

Parameters

Table 13–6 describes the parameters available in the GET_SESSION_STATE function.

Table 13–6 GET_SESSION_STATE Parameters

Parameter	Description
p_item	Case insensitive name of the item for which you wish to fetch session state.

Example

```
DECLARE
      l_item_value  VARCHAR2(255);
BEGIN
      l_item_value := HTMLDB_UTIL.GET_SESSION_STATE('my_item');
END;
```

PUBLIC_CHECK_AUTHORIZATION Function

Given the name of a security scheme, this function determines if the current user passes the security check.

Syntax

```
HTMLDB_UTIL.PUBLIC_CHECK_AUTHORIZATION (
    p_security_scheme    IN    VARCHAR2)
    RETURN BOOLEAN;
```

Parameters

Table 13–7 describes the parameters available in the PUBLIC_CHECK_AUTHORIZATION function.

Table 13–7 PUBLIC_CHECK_AUTHORIZATION Parameters

Parameter	Description
p_security_name	Name of the security scheme that determines if the user passes the security check.

Example

```
DECLARE
      l_check_security  boolean;
BEGIN
      l_check_security := HTMLDB_UTIL.PUBLIC_CHECK_AUTHORIZATION('my_auth_
scheme');
END;
```

REMOVE_PREFERENCE Procedure

This function returns removes the preference for the supplied user.

Syntax

```
HTMLDB_UTIL.REMOVE_PREFERENCE(
    p_preference    IN    VARCHAR2 DEFAULT NULL,
    p_user          IN    VARCHAR2 DEFAULT V('USER'));
```

Parameters

Table 13–8 describes the parameters available in the `REMOVE_PREFERENCE` procedure.

Table 13–8 REMOVE_PREFERENCE Parameters

Parameter	Description
`p_preference`	Name of the preference to remove.
`p_user`	User for whom the preference is for.

Example

```
BEGIN
       HTMLDB_UTIL.REMOVE_PREFERENCE(
                    p_preference => 'default_view',
                    p_user       => :APP_USER);
END;
```

REMOVE_SORT_PREFERENCES Procedure

This procedure removes the user's column heading sorting preference value.

Syntax

```
HTMLDB_UTIL.REMOVE_SORT_PREFERENCES (
    p_user  IN VARCHAR2 DEFAULT V('USER'));
```

Parameters

Table 13–9 describes the parameters available in the REMOVE_SORT_PREFERENCES procedure.

Table 13–9 REMOVE_SORT_PREFERENCES Parameters

Parameter	Description
p_user	User for whom sorting preference will be removed.

Example

```
BEGIN
      HTMLDB_UTIL.REMOVE_SORT_PREFERENCES(:APP_USER);
END;
```

RESET_AUTHORIZATIONS Procedure

To increase performance, Oracle HTML DB caches security checks. You can use this procedure to undo caching thus requiring all security checks be revalidated for the current user. Use this procedure if you wish users to have the ability to change their responsibilities (their authorization profile) within your application.

Syntax

```
HTMLDB_UTIL.RESET_AUTHORIZATIONS;
```

Example

```
BEGIN
HTMLDB_UTIL.RESET_AUTHORIZATIONS;
END;
```

SET_PREFERENCE Procedure

This procedure sets a preference that will persist beyond the user's current session.

Syntax

```
HTMLDB_UTIL.SET_PREFERENCE (
    p_preference   IN    VARCHAR2 DEFAULT NULL,
    p_value        IN    VARCHAR2 DEFAULT NULL,
    p_user         IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 13–10 describes the parameters available in the SET_PREFERENCE procedure.

Table 13–10 SET_PREFERENCE Parameters

Parameter	Description
p_preference	Name of the preference (case sensitive).
p_value	Value of the preference.
p_user	User for whom the preference is being set.

Example

```
BEGIN
       HTMLDB_UTIL.SET_PREFERENCE(
             p_preference => 'default_view',
             p_value      => 'WEEKLY',
             p_user       => :APP_USER);
END;
```

SET_SESSION_STATE Procedure

This procedure sets session state for a current Oracle HTML DB session.

Syntax

```
HTMLDB_UTIL.SET_SESSION_STATE (
    p_name     IN    VARCHAR2 DEFAULT NULL,
    p_value    IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 13–11 describes the parameters available in the SET_SESSION_STATE procedure.

Table 13–11 SET_SESSION_STATE Parameters

Parameter	Description
p_name	Name of the application or page level item for which you are setting sessions state.
p_value	Value of session state to set.

Example

```
BEGIN
HTMLDB_UTIL.SET_SESSION_STATE('my_item','myvalue');
END;
```

STRING_TO_TABLE Function

Given a string, this function returns a PL/SQL array of type HTMLDB_APPLICATION_GLOBAL.VC_ARR2. This array is a VARCHAR2(32767) table.

Syntax

```
HTMLDB_UTIL.STRING_TO_TABLE (
    p_string       IN VARCHAR2,
    p_separator    IN VARCHAR2 DEFAULT ':')
    RETURN HTMLDB_APPLICATION_GLOBAL.VC_ARR2;
```

Parameters

Table 13–12 describes the parameters available in the STRING_TO_TABLE function.

Table 13–12 STRING_TO_TABLE Parameters

Parameter	Description
p_string	String to be converted into a PL/SQL table of type HTMLDB_APPLICATION_GLOBAL.VC_ARR2.
p_separator	String separator. The default is a colon.

Example

```
DECLARE
       l_vc_arr2    HTMLDB_APPLICATION_GLOBAL.VC_ARR2;
BEGIN
       l_vc_arr2 := HTMLDB_UTIL.STRING_TO_TABLE('One:Two:Three');
```

```
      FOR z IN 1..l_vc_arr2.count LOOP
            htp.p(l_vc_arr2(z));
      END LOOP;
END;
```

TABLE_TO_STRING Function

Given a a PL/SQL table of type HTMLDB_APPLICATION_GLOBAL.VC_ARR2, this function returns a delimited string separated by the supplied separator, or by the default separator, a colon (:).

Syntax

```
HTMLDB_UTIL.TABLE_TO_STRING (
    p_table       IN     HTMLDB_APPLICATION_GLOBAL.VC_ARR2,
    p_string      IN     VARCHAR2 DEFAULT ':')
    RETURN VARCHAR2;
```

Parameters

Table 13–13 describes the parameters available in the TABLE_TO_STRING function.

Table 13–13 TABLE_TO_STRING Parameters

Parameter	Description
p_string	String separator. Default separator is a colon (:).
p_table	PL/SQL table that is to be converted into a delimited string.

Example

```
DECLARE
      l_string     VARCHAR2(255);
      l_vc_arr2    HTMLDB_APPLICATION_GLOBAL.VC_ARR2;
BEGIN
      l_vc_arr2 := HTMLDB_UTIL.STRING_TO_TABLE('One:Two:Three');

      l_string := HTMLDB_UTIL.TABLE_TO_STRING(l_vc_arr2);
END;
```

URL_ENCODE Function

This function encodes (into HEX) all special characters that include spaces, question marks, ampersands, and so on.

Syntax

```
HTMLDB_UTIL.URL_ENCODE (
    p_url   IN    VARCHAR2)
    RETURN VARCHAR2;
```

Parameters

Table 13–14 describes the parameters available in the URL_ENCODE function.

Table 13–14 URL_ENCODE Parameters

Parameter	Description
p_string	The string you would like to have encoded.

Example

```
DECLARE
      l_url  VARCHAR2(255);
BEGIN
      l_url := HTMLDB_UTIL.URL_ENCODE('http://www.myurl.com?id=1&cat=foo');
END;
```

HTMLDB_ITEM

You can use the HTMLDB_ITEM package to create form elements dynamically based on a SQL query instead of creating individual items page by page.

Topics in this section include:

- CHECKBOX Function
- DATE_POPUP Function
- HIDDEN Function
- MD5_CHECKSUM Function
- MD5_HIDDEN Function

CHECKBOX Function

This function creates check boxes.

Syntax

```
HTMLDB_ITEM.CHECKBOX(
    p_idx                       IN    NUMBER,
    p_value                     IN    VARCHAR2 DEFAULT,
    p_attributes                IN    VARCHAR2 DEFAULT,
    p_checked_values            IN    VARCHAR2 DEFAULT,
    p_checked_values_delimitor  IN    VARCHAR2 DEFAULT)
    RETURN VARCHAR2;
```

Parameters

Table 13–15 describes the parameters available in the `CHECKBOX` function.

Table 13–15 CHECKBOX Parameters

Parameter	Description
p_idx	Number which determines which HTMLDB_APPLICATION global will be used. Valid range of values is 1 to 50. For example 1 creates F01 and 2 creates F02.
p_value	Value of a check box, hidden field, or input form item.
p_attributes	Controls HTML tag attributes (such as disabled).
p_checked_values	Values to be checked by default.
p_checked_values_delimitor	Delimits the values in the previous parameter, p_checked_values.

Examples of Default Check Box Behavior

The following example demonstrates how to create a selected check box for each employee in the emp table.

```
SELECT HTMLDB_ITEM.CHECKBOX(1,empno,'CHECKED') " ",
       ename,
       job
FROM   emp
ORDER BY 1
```

The next example demonstrates how to have all check boxes for employees display without being selected.

```
SELECT HTMLDB_ITEM.CHECKBOX(1,empno) " ",
       ename,
       job
FROM   emp
ORDER BY 1
```

The next example demonstrates how to select the check boxes for employees who work in department 10.

```
SELECT HTMLDB_ITEM.CHECKBOX(1,empno,DECODE(deptno,10,'CHECKED',null)) " ",
       ename,
       job
FROM   emp
ORDER BY 1
```

The next example demonstrates how to select the check boxes for employees who work in department 10 or department 20.

```
SELECT HTMLDB_ITEM.CHECKBOX(1,deptno,NULL,'10:20',':') " ",
       ename,
       job
FROM   emp
ORDER BY 1
```

Creating a On-Submit Process

If you are using check boxes in your application, you might need to create an On Submit process to perform a specific type of action on the selected rows. For example, you could have a Delete button that utilizes the following logic:

```
SELECT HTMLDB_ITEM.CHECKBOX(1,empno) " ",
       ename,
       job
FROM   emp
ORDER  by 1
```

Consider the following sample on-submit process:

```
FOR I in 1..HTMLDB_APPLICATION.G_F01.COUNT LOOP
    DELETE FROM emp WHERE empno = to_number(HTMLDB_APPLICATION.G_F01(i));
END LOOP;
```

DATE_POPUP Function

Use this function with forms that include date fields. DATE_POPUP dynamically generates a date field that has popup calendar button.

Syntax

```
HTMLDB_ITEM.DATE_POPUP(
    p_idx          IN    NUMBER,
    p_row          IN    NUMBER,
    p_value        IN    VARCHAR2 DEFAULT,
    p_date_format  IN    DATE DEFAULT,
    p_size         IN    NUMBER DEFAULT,
    p_maxlength    IN    NUMBER DEFAULT,
    p_attributes   IN    VARCHAR2 DEFAULT)
    RETURN VARCHAR2;
```

Parameters

Table 13–16 describes the parameters available in the DATE_POPUP function.

Table 13–16 DATE_POPUP Parameters

Parameter	Description
p_idx	Number which determines which HTMLDB_APPLICATION global will be used.Valid range of values is 1 to 50. For example 1 creates F01 and 2 creates F02.
p_value	Value of a field item.
p_date_format	Valid database date format.
p_size	Controls HTML tag attributes (such as disabled).
p_maxlength	Determine the maximum number of enterable characters. Becomes the maxlength attribute of the <input > HTML tag.
p_attributes	Extra HTML parameters you wish to add.

> **See Also:** *Oracle Database SQL Reference* for more information on the TO_CHAR or TO_DATE functions

Example

The following example demonstrates how to use HTMLDB_ITEM.DATE_POPUP to create popup calendar buttons for the hiredate column.

```
SELECT
  empno,
  HTMLDB_ITEM.HIDDEN(1,empno)||
  HTMLDB_ITEM.TEXT(2,ename) ename,
  HTMLDB_ITEM.TEXT(3,job) job,
  mgr,
  HTMLDB_ITEM.DATE_POPUP(4,rownum,hiredate,'dd-mon-yyyy') hd,
  HTMLDB_ITEM.TEXT(5,sal) sal,
  HTMLDB_ITEM.TEXT(6,comm) comm,
  deptno
FROM emp
ORDER BY 1
```

HIDDEN Function

This function dynamically generates hidden form items.

Syntax

```
HTMLDB_ITEM.HIDDEN(
    p_idx     IN    NUMBER,
    p_value   IN    VARCHAR2 DEFAULT)
    RETURN VARCHAR2;
```

Parameters

Table 13–17 describes the parameters available in the HIDDEN function.

Table 13–17 HIDDEN Parameters

Parameter	Description
p_idx	Number to identify the item you wish to generate. The number will determine which G_FXX global is populated. **See Also:** "HTMLDB_APPLICATION" on page 13-40
p_value	Value of the hidden input form item.

Example

Typically, the primary key of a table is stored as a hidden column and used for subsequent update processing. Consider the following sample SLQ query:

```
SELECT
  empno,
  HTMLDB_ITEM.HIDDEN(1,empno)||
  HTMLDB_ITEM.TEXT(2,ename) ename,
  HTMLDB_ITEM.TEXT(3,job) job,
  mgr,
  HTMLDB_ITEM.DATE_POPUP(4,rownum,hiredate,'dd-mon-yyyy') hiredate,
  HTMLDB_ITEM.TEXT(5,sal) sal,
  HTMLDB_ITEM.TEXT(6,comm) comm,
  deptno
FROM emp
ORDER BY 1
```

The previous query could use the following page process to process the results:

```
BEGIN
```

```
  FOR i IN 1..HTMLDB_APPLICATION.G_F01.COUNT LOOP
    UPDATE emp
    SET
      ename=HTMLDB_APPLICATION.G_F02(i),
      job=HTMLDB_APPLICATION.G_F03(i),
      hiredate=to_date(HTMLDB_APPLICATION.G_F04(i),'dd-mon-yyyy'),
      sal=HTMLDB_APPLICATION.G_F05(i),
      comm=HTMLDB_APPLICATION.G_F06(i)
    WHERE empno=to_number(HTMLDB_APPLICATION.G_F01(i));
  END LOOP;
END;
```

Note that the G_F01 column (which corresponds to the hidden EMPNO) is used as the key to update each row.

MD5_CHECKSUM Function

This function passes values to HTMLDB_ITEM.MULTI_ROW_UPDATE and is used for lost update detection. Lost update detection ensures data integrity in applications where data can be accessed concurrently.

Syntax

```
HTMLDB_ITEM.MD5_CHECKSUM(
    p_value01   IN    VARCHAR2 DEFAULT,
    p_value02   IN    VARCHAR2 DEFAULT,
    p_value03   IN    VARCHAR2 DEFAULT,
    ...
    p_value50   IN    VARCHAR2 DEFAULT,
    p_col_sep   IN    VARCHAR2 DEFAULT,
    RETURN VARCHAR2;
```

Parameters

Table 13–19 describes the parameters available in the MD5_CHECKSUM function.

Table 13–18 MD5_HIDDEN Parameters

Parameter	Description
p_value01	Fifty available inputs. Parameters that are not supplied default to null.
...	
p_value50	

Table 13–18 MD5_HIDDEN Parameters

Parameter	Description
p_col_sep	String used to separate p_value inputs. Defaults to the pipe symbol (|).

Example

```
SELECT HTMLDB_ITEM.MD5_CHECKSUM(ename,job,sal)
FROM emp
```

MD5_HIDDEN Function

This function is used for lost update detection which ensures data integrity in applications where data can be accessed concurrently.

This function produces a hidden form field and includes 50 inputs. HTMLDB_ITEM.MD5_HIDDEN also produces an MD5 checksum using the Oracle database DBMS_OBFUSCATION_TOOLKIT:

```
UTL_RAW.CAST_TO_RAW(DBMS_OBFUSCATION_TOOLKIT.MD5())
```

An MD5 checksum provides data integrity through hashing and sequencing to assure that data is not altered or stolen as it is transmitted over a network

Syntax

```
HTMLDB_ITEM.MD5_HIDDEN(
    p_idx       IN    NUMBER,
    p_value01   IN    VARCHAR2 DEFAULT,
    p_value02   IN    VARCHAR2 DEFAULT,
    p_value03   IN    VARCHAR2 DEFAULT,
    ...
    p_value50   IN    VARCHAR2 DEFAULT,
    p_col_sep   IN    VARCHAR2 DEFAULT,
    RETURN VARCHAR2;
```

Parameters

Table 13–19 describes the parameters available in the MD5_HIDDEN function.

Table 13–19 MD5_HIDDEN Parameters

Parameter	Description
p_idx	Indicates the form element to be generated. For example, 1 equals F01 and 2 equals F02. Typically the p_idx parameter is constant for a given column.
p_value01	Fifty available inputs. Parameters not supplied default to null.
...	
p_value50	
p_col_sep	String used to separate p_value inputs. Defaults to the pipe symbol (|).

Example

p_idx specifies the FXX form element to be generated. In the following example, 7 generates F07. Also note that an HTML hidden form element will be generated.

```
SELECT HTMLDB_ITEM.MD5_HIDDEN(7,ename,job,sal), ename, job, sal FROM emp
```

MULTI_ROW_UPDATE Procedure

Use this procedure within a Multi Row Update process type. This procedure takes a string containing a multiple row update definition in the following format:

```
OWNER:TABLE:pk_column1,pk_idx:pk_column2,pk_idx2|col,idx:col,idx...
```

Syntax

```
HTMLDB_ITEM.MULTI_ROW_UPDATE(
    p_mru_string    IN    VARCHAR2 DEFAULT)
    RETURN VARCHAR2;
```

Example

To use this procedure indirectly within application level process, you need to create a query to generate a form of database data. The following example demonstrates how to create a multiple row update on the emp table.

```
SELECT
empno,
HTMLDB_ITEM.HIDDEN(1,empno),
HTMLDB_ITEM.HIDDEN(2,deptno),
```

```
HTMLDB_ITEM.TEXT(3,ename),
HTMLDB_ITEM.SELECT_LIST_FROM_QUERY(4,job,'SELECT DISTINCT job FROM emp'),
HTMLDB_ITEM.TEXT(5,sal),
HTMLDB_ITEM.TEXT(7,comm),
HTMLDB_ITEM.MD5_CHECKSUM(ename,job,sal,comm),
deptno
FROM emp
WHERE deptno = 20
```

Note the call to HTMLDB_ITEM.MD5_CHECKSUM instead of HTMLDB_ITEM.MD5_HIDDEN. Since HTMLDB_ITEM.MULTI_ROW_UPDATE gets the checksum from HTMLDB_APPLICATION.G_FCS, you need to call HTMLDB_ITEM.MD5_CHECKSUM in order to populate HTMLDB_APPLICATION.G_FCS when the page is submitted. Additionally, the columns in HTMLDB_ITEM.MD5_CHECKSUM must be in the same order those in the MULTI_ROW_UPDATE process. These updates can then processed (or applied to the database) using an after submit page process of Multi Row Update in a string similar to the following:

```
SCOTT:emp:empno,1:deptno,2|ename,3:job,4:sal,5:comm,7:,:,:,:,
```

SELECT_LIST Function

This function dynamically generates a static select list. Similar to other functions available in the HTMLDB_ITEM package, these select list functions are designed to generate forms with F01 to F50 form array elements.

Syntax

```
HTMLDB_ITEM.SELECT_LIST(
    p_idx           IN   NUMBER,
    p_value         IN   VARCHAR2 DEFAULT,
    p_list_values   IN   VARCHAR2 DEFAULT,
    p_attributes    IN   VARCHAR2 DEFAULT,
    p_show_null     IN   VARCHAR2 DEFAULT,
    p_null_value    IN   VARCHAR2 DEFAULT,
    p_null_text     IN   VARCHAR2 DEFAULT)
    RETURN VARCHAR2;
```

Parameters

Table 13–20 describes the parameters available in the SELECT_LIST function.

Table 13–20 SELECT_LIST Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically the P_IDX parameter is constant for a given column.
p_value	Current value. This value should be a value in the P_LIST_VALUES parameter.
p_list_values	List of static values separated by commas. Display values and return values are separated by semicolons. Note that this is only available in the SELECT_LIST function.
p_attributes	Extra HTML parameters you wish to add.
p_show_null	Extra select option to enable the NULL selection. Range of values is YES and NO.
p_null_value	Value to be returned when a user selects the null option. Only relevant when P_SHOW_NULL equals YES.

Example

The following example demonstrates a static select list that displays Yes, returns Y, defaults to Y, and generates a F01 form item.

```
SELECT HTMLDB_ITEM.SELECT_LIST(1,'Y','Yes;Y,No;N')
FROM emp
```

SELECT_LIST_FROM_LOV Function

This function dynamically generates select lists from a shared list of values (LOV). Similar to other functions available in the HTMLDB_ITEM package, these select list functions are designed to generate forms with F01 to F50 form array elements.

Syntax

```
HTMLDB_ITEM.SELECT_LIST_FROM_LOV(
    p_idx           IN   NUMBER,
    p_value         IN   VARCHAR2 DEFAULT,
    p_lov           IN   VARCHAR2,
    p_attributes    IN   VARCHAR2 DEFAULT,
    p_show_null     IN   VARCHAR2 DEFAULT,
    p_null_value    IN   VARCHAR2 DEFAULT,
    p_null_text     IN   VARCHAR2 DEFAULT)
    RETURN VARCHAR2;
```

Parameters

Table 13–21 describes the parameters available in the SELECT_LIST_FROM_LOV function.

Table 13–21 SELECT_LIST_FROM_LOV Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically the p_idx parameter is constant for a given column.
p_value	Current value. This value should be a value in the p_list_values parameter.
p_lov	Text name of a flow list of values. This list of values must be defined in your flow. This parameter is used only by the select_list_from_lov function.
p_attributes	Extra HTML parameters you wish to add.
p_show_null	Extra select option to enable the NULL selection. Range of values is YES and NO.
p_null_value	Value to be returned when a user selects the null option. Only relevant when p_show_null equals YES.
p_null_text	Value to be displayed when a user selects the null option. Only relevant when p_show_null equals YES.

Example

The following demonstrates a select list based on a LOV defined in the application.

```
SELECT HTMLDB_ITEM.SELECT_LIST_FROM_LOV(2,job,'JOB_FLOW_LOV')
FROM emp
```

SELECT_LIST_FROM_LOV_XL Function

This function dynamically generates very large select lists (greater than 32K) from a shared list of values (LOV). Similar to other functions available in the HTMLDB_ITEM package, these select list functions are designed to generate forms with F01 to F50 form array elements.

Syntax

```
HTMLDB_ITEM.SELECT_LIST_FROM_LOV_XL(
```

```
p_idx           IN   NUMBER,
p_value         IN   VARCHAR2 DEFAULT,
p_lov           IN   VARCHAR2,
p_attributes    IN   VARCHAR2 DEFAULT,
p_show_null     IN   VARCHAR2 DEFAULT,
p_null_value    IN   VARCHAR2 DEFAULT,
p_null_text     IN   VARCHAR2 DEFAULT)
RETURN CLOB;
```

Parameters

Table 13–22 describes the parameters available in the SELECT_LIST_FROM_LOV_XL function.

Table 13–22 SELECT_LIST_FROM_LOV_XL Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically the p_idx parameter is constant for a given column.
p_value	Current value. This value should be a value in the p_list_values parameter.
p_lov	Text name of a flow list of values. This list of values must be defined in your flow. This parameter is used only by the select_list_from_lov function.
p_attributes	Extra HTML parameters you wish to add.
p_show_null	Extra select option to enable the NULL selection. Range of values is YES and NO.
p_null_value	Value to be returned when a user selects the null option. Only relevant when p_show_null equals YES.
p_null_text	Value to be displayed when a user selects the null option. Only relevant when p_show_null equals YES.

Example

The following demonstrates a select list based on a LOV defined in the application.

```
SELECT HTMLDB_ITEM.SELECT_LIST_FROM_LOV_XL(2,job,'JOB_FLOW_LOV')
FROM emp
```

SELECT_LIST_FROM_QUERY Function

This function dynamically generates a select list from a query. Similar to other functions available in the HTMLDB_ITEM package, these select list functions are designed to generate forms with F01 to F50 form array elements.

Syntax

```
HTMLDB_ITEM.SELECT_LIST_FROM_QUERY(
    p_idx           IN    NUMBER,
    p_value         IN    VARCHAR2 DEFAULT,
    p_query         IN    VARCHAR2,
    p_attributes    IN    VARCHAR2 DEFAULT,
    p_show_null     IN    VARCHAR2 DEFAULT,
    p_null_value    IN    VARCHAR2 DEFAULT,
    p_null_text     IN    VARCHAR2 DEFAULT)
    RETURN VARCHAR2;
```

Parameters

Table 13–23 describes the parameters available in the SELECT_LIST_FROM_QUERY function.

Table 13–23 SELECT_LIST_FROM_QUERY Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically the p_idx parameter is constant for a given column.
p_value	Current value. This value should be a value in the p_list_values parameter.
p_query	SQL query that is expected to select two columns, a display column, and a return column. For example: `SELECT dname, deptno FROM dept` Note that this is used only by the SELECT_LIST_FROM_QUERY function.
p_attributes	Extra HTML parameters you wish to add.
p_show_null	Extra select option to enable the NULL selection. Range of values is YES and NO.
p_null_value	Value to be returned when a user selects the null option. Only relevant when p_show_null equals YES.

Table 13–23 SELECT_LIST_FROM_QUERY Parameters

Parameter	Description
p_null_text	Value to be displayed when a user selects the null option. Only relevant when p_show_null equals YES.

Example

The following demonstrates a select list based on a SQL query.

```
SELECT HTMLDB_ITEM.SELECT_LIST_FROM_QUERY(3,job,'SELECT DISTINCT job FROM emp')
FROM emp
```

SELECT_LIST_FROM_QUERY_XL Function

This function dynamically generates very large select lists (greater than 32K) from a query. Similar to other functions available in the HTMLDB_ITEM package, these select list functions are designed to generate forms with F01 to F50 form array elements.

Syntax

```
HTMLDB_ITEM.SELECT_LIST_FROM_QUERY_XL(
    p_idx           IN    NUMBER,
    p_value         IN    VARCHAR2 DEFAULT,
    p_query         IN    VARCHAR2,
    p_attributes    IN    VARCHAR2 DEFAULT,
    p_show_null     IN    VARCHAR2 DEFAULT,
    p_null_value    IN    VARCHAR2 DEFAULT,
    p_null_text     IN    VARCHAR2 DEFAULT)
    RETURN CLOB;
```

Parameters

Table 13–24 describes the parameters available in the SELECT_LIST_FROM_QUERY_XL function.

Table 13–24 SELECT_LIST_FROM_QUERY_XL Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically the p_idx parameter is constant for a given column.

Table 13–24 SELECT_LIST_FROM_QUERY_XL Parameters

Parameter	Description
p_value	Current value. This value should be a value in the p_list_values parameter.
p_query	SQL query that is expected to select two columns, a display column, and a return column. For example: `SELECT dname, deptno FROM dept` Note that this is used only by the SELECT_LIST_FROM_QUERY_XL function.
p_attributes	Extra HTML parameters you wish to add.
p_show_null	Extra select option to enable the NULL selection. Range of values is YES and NO.
p_null_value	Value to be returned when a user selects the null option. Only relevant when p_show_null equals YES.
p_null_text	Value to be displayed when a user selects the null option. Only relevant when p_show_null equals YES.

Example

The following demonstrates a select list based on a SQL query.

```
SELECT HTMLDB_ITEM.SELECT_LIST_FROM_QUERY_XL(3,job,'SELECT DISTINCT job FROM
emp')
FROM emp
```

TEXT Function

This function generates text fields (or text input form items) from a SQL query.

Syntax

```
HTMLDB_ITEM.TEXT(
    p_idx         IN    NUMBER,
    p_value       IN    VARCHAR2 DEFAULT NULL,
    p_size        IN    NUMBER DEFAULT NULL,
    p_maxlength   IN    NUMBER DEFAULT NULL,
    p_attributes  IN    VARCHAR2 DEFAULT NULL,
    p_item_id     IN    VARCHAR2 DEFAULT NULL,
    p_item_label  IN    VARCHAR2 DEFAULT NULL)
```

Parameters

Table 13–25 describes the parameters available in the TEXT function.

Table 13–25 TEXT Parameters

Parameter	Description
p_idx	Number to identify the item you wish to generate. The number will determine which G_FXX global is populated. **See Also:** "HTMLDB_APPLICATION" on page 13-40
p_value	Value of a text field item.
p_size	Controls HTML tag attributes (such as disabled).
p_maxlength	Maximum number of characters that can be entered in the text box.
p_attributes	Extra HTML parameters you wish to add.
p_item_id	HTML attribute ID for the <input> tag.
p_item_label	Label of the text field item.

Example

The following sample query demonstrates how to generate one update field for each row. Note that the ename, sal, and comm columns use the HTMLDB_ITEM.TEXT function to generate an HTML text field for each row. Also, notice that each item in the query is passed an unique p_idx parameter to ensure that each column is stored in its own array.

```
SELECT
  empno,
  HTMLDB_ITEM.HIDDEN(1,empno)||
  HTMLDB_ITEM.TEXT(2,ename) ename,
  HTMLDB_ITEM.TEXT(3,job) job,
  mgr,
  HTMLDB_ITEM.DATE_POPUP(4,rownum,hiredate,'dd-mon-yyyy') hiredate,
  HTMLDB_ITEM.TEXT(5,sal) sal,
  HTMLDB_ITEM.TEXT(6,comm) comm,
  deptno
FROM emp
ORDER BY 1
```

TEXT_FROM_LOV Function

This function returns the display value of a LOV given its value.

Syntax

```
HTMLDB_ITEM.TEXT_FROM_LOV (
    p_value       IN    VARCHAR2 DEFAULT NULL,
    p_lov         IN    VARCHAR2,
    p_null_text   IN    VARCHAR2 DEFAULT '%')
    RETURN VARCHAR2;
```

Parameters

Table 13–26 describes the parameters available in the TEXT_FROM_LOV function.

Table 13–26 TEXT_FROM_LOV Parameters

Parameter	Description
p_value	Display value of the LOV you are retrieving.
p_lov	Name of the LOV in your application.
p_null_text	Text to display if the value is null.

Example

Suppose you have an LOV called DEPARTMENTS_LOV as shown in the following example:

```
SELECT dname, deptno FROM dept;
```

Next, assume you have a SQL Query region and you wish to query the emp table. However, instead of displaying the deptno column (which contains numbers), you wish to show the department name. You can accomplish this by using HTMLDB_ITEM.TEXT_FROM_LOV function. For example:

```
SELECT ename, job, sal, comm, HTMLDB_ITEM.TEXT_FROM_LOV(deptno,'DEPARTMENTS_
LOV') d FROM emp;
```

RADIOGROUP Function

This function generates a radio group from a SQL query.

Syntax

```
HTMLDB_ITEM.RADIOGROUP(
    p_idx              IN    NUMBER,
    p_value            IN    VARCHAR2 DEFAULT,
    p_selected_value   IN    VARCHAR2 DEFAULT,
    p_display          IN    VARCHAR2 DEFAULT,
    p_attributes       IN    VARCHAR2 DEFAULT,
    p_onblur           IN    VARCHAR2 DEFAULT,
    p_onchange         IN    VARCHAR2 DEFAULT,
    p_onfocus          IN    VARCHAR2 DEFAULT,)
    RETURN VARCHAR2;
```

Parameters

Table 13–27 describes the parameters available in the RADIOGROUP function.

Table 13–27 RADIOGROUP Parameters

Parameter	Description
p_idx	Number which determines which HTMLDB_APPLICATION global will be used. Valid range of values is 1 to 50.For example 1 creates F01 and 2 creates F02.
p_value	Value of the radio group.
p_selected_value	Value that should be "on", or selected.
p_display	Text to display next to the radio option.
p_attributes	Extra HTML parameters you wish to add.
p_onblur	JavaScript to execute in the onBlur event.
p_onchange	JavaScript to execute in the onChange event.
p_onfocus	JavaScript to execute in the onFocus event.

Example

The following example demonstrates how to select department 20 from the emp table as a default in a radio group.

```
SELECT HTMLDB_ITEM.CHECKBOX(1,deptno,'20',dname) dt
FROM   dept
ORDER  BY 1
```

POPUP_FROM_LOV Function

This function generates an HTML popup select list from an application list of values (LOV). Like other available functions in the HTMLDB_ITEM package, POPUP_FROM_LOV is designed to generate forms with F01 to F50 form array elements.

Syntax

```
HTMLDB_ITEM.POPUP_FROM_LOV(

    p_idx              IN    NUMBER,
    p_value            IN    VARCHAR2 DEFAULT,
    p_lov_name         IN    VARCHAR2,
    p_width            IN    VARCHAR2 DEFAULT,
    p_max_length       IN    VARCHAR2 DEFAULT,
    p_form_index       IN    VARCHAR2 DEFAULT,
    p_escape_html      IN    VARCHAR2 DEFAULT,
    p_max_elements     IN    VARCHAR2 DEFAULT,
    p_attributes       IN    VARCHAR2 DEFAULT,
    p_ok_to_query      IN    VARCHAR2 DEFAULT,
    p_item_id          IN    VARCHAR2 DEFAULT NULL,
    p_item_label       IN    VARCHAR2 DEFAULT NULL)
    RETURN VARCHAR2;
```

Parameters

Table 13–28 describes the some parameters in the POPUP_FROM_LOV function.

Table 13–28 POPUP_FROM_LOV Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, p_idx is a constant for a given column.
p_value	Form element current value. This value should be one of the values in the p_lov_name parameter.
p_lov_name	Named LOV used for this popup.
p_width	Width of the text box.
p_max_length	Maximum number of characters that can be entered in the text box.

Table 13–28 POPUP_FROM_LOV Parameters

Parameter	Description
p_form_index	HTML form on the page in which an item is contained. Defaults to 0 and rarely used. Only use this parameter when it is necessary to embed a custom form in your page template (such as a search field which posts to a different Web site). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle HTML DB must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV which passes a value back to a form element.
p_escape_html	Replacements for special characters that require an escaped equivalent. ■ < for < ■ > for > ■ & for & This parameter is useful if you know your query will return illegal HTML.
p_max_elements	Limit on the number of rows that can be returned by your query. Limits the performance impact of user searches. By entering a value in this parameter, you force the user to search for a more narrow set of results.
p_attributes	Additional HTML attributes to use for the form item.
p_ok_to_query	Range of values is YES and NO. If YES, a popup returns first set of rows for the LOV. If NO, a search is initiated to return rows.
p_item_id	ID attribute of the form element.
p_item_label	Invisible label created for the item.

Example

The following example demonstrates a sample query the generates a popup from a LOV named DEPT.

```
SELECT HTMLDB_ITEM.POPUP_FROM_LOV (1,deptno,'DEPT_LOV') dt
FROM emp
```

POPUP_FROM_QUERY Function

This function generates an HTML popup select list from a query. Like other available functions in the HTMLDB_ITEM package, POPUP_FROM_QUERY is designed to generate forms with F01 to F50 form array elements.

Syntax

```
HTMLDB_ITEM.POPUP_FROM_QUERY(

    p_idx              IN    NUMBER,
    p_value            IN    VARCHAR2 DEFAULT,
    p_lov_query        IN    VARCHAR2,
    p_width            IN    VARCHAR2 DEFAULT,
    p_max_length       IN    VARCHAR2 DEFAULT,
    p_form_index       IN    VARCHAR2 DEFAULT,
    p_escape_html      IN    VARCHAR2 DEFAULT,
    p_max_elements     IN    VARCHAR2 DEFAULT,
    p_attributes       IN    VARCHAR2 DEFAULT,
    p_ok_to_query      IN    VARCHAR2 DEFAULT,
    p_item_id          IN    VARCHAR2 DEFAULT NULL,
    p_item_label       IN    VARCHAR2 DEFAULT NULL)
    RETURN VARCHAR2;
```

Parameters

Table 13–29 describes the parameters in the POPUP_FROM_QUERY function.

Table 13–29 POPUP_FROM_QUERY Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, p_idx is a constant for a given column.
p_value	Form element current value. This value should be one of the values in the p_lov_query parameter.
p_lov_query	SQL query that is expected to select two columns (a display column and a return column). For example: `SELECT dname, deptno FROM dept`
p_width	Width of the text box.
p_max_length	Maximum number of characters that can be entered in the text box.

Table 13–29 POPUP_FROM_QUERY Parameters

Parameter	Description
p_form_index	HTML form on the page in which an item is contained. Defaults to 0 and rarely used. Only use this parameter when it is necessary to embed a custom form in your page template (such as a search field which posts to a different Web site). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle HTML DB must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV which passes a value back to a form element.
p_escape_html	Replacements for special characters that require an escaped equivalent. ■ < for < ■ > for > ■ & for & This parameter is useful if you know your query will return illegal HTML.
p_max_elements	Limit on the number of rows that can be returned by your query. Limits the performance impact of user searches. By entering a value in this parameter, you force the user to search for a more narrow set of results.
p_attributes	Additional HTML attributes to use for the form item.
p_ok_to_query	Range of values is YES and NO. If YES, a popup returns first set of rows for the LOV. If NO, a search is initiated to return rows.
p_item_id	ID attribute of the form element.
p_item_label	Invisible label created for the item.

Example

The following example demonstrates a sample query the generates a popup select list from the emp table.

```
SELECT HTMLDB_ITEM.POPUP_FROM_QUERY (1,deptno,'SELECT dname, deptno FROM dept')
dt
FROM emp
```

POPUPKEY_FROM_LOV Function

This function generates a popup key select list from a shared list of values (LOV). Like other available functions in the HTMLDB_ITEM package, POPUPKEY_FROM_LOV is designed to generate forms with F01 to F50 form array elements.

Syntax

```
HTMLDB_ITEM.POPUPKEY_FROM_LOV(
    p_idx              IN    NUMBER,
    p_value            IN    VARCHAR2 DEFAULT,
    p_lov_name         IN    VARCHAR2,
    p_width            IN    VARCHAR2 DEFAULT,
    p_max_length       IN    VARCHAR2 DEFAULT,
    p_form_index       IN    VARCHAR2 DEFAULT,
    p_escape_html      IN    VARCHAR2 DEFAULT,
    p_max_elements     IN    VARCHAR2 DEFAULT,
    p_attributes       IN    VARCHAR2 DEFAULT,
    p_ok_to_query      IN    VARCHAR2 DEFAULT,
    RETURN VARCHAR2;
```

Although the text field associated with the popup displays in the first column in the LOV query, the actual value is specified in the second column in the query.

Parameters

Table 13–30 describes the some parameters in the POPUPKEY_FROM_LOV function.

Table 13–30 POPUPKEY_FROM_LOV Parameters

Parameter	Description
p_idx	Identifies a form element name. For example, 1 equals F01 and 2 equals F02. Typically, p_idx is a constant for a given column Because of the behavior of POPUPKEY_FROM_QUERY, the next index value should be p_idx + 1. For example: `SELECT HTMLDB_ITEM.POPUPKEY_FROM_LOV (1,deptno,'DEPT') dt, HTMLDB_ITEM.HIDDEN(3,empno) eno`
p_value	Indicates the current value. This value should be one of the values in the P_LOV_NAME parameter.
p_lov_name	Identifies a named LOV used for this popup.
p_width	Width of the text box.
p_max_length	Maximum number of characters that can be entered in the text box.

Table 13–30 POPUPKEY_FROM_LOV Parameters

Parameter	Description
p_form_index	HTML form on the page in which an item is contained. Defaults to 0 and rarely used. Only use this parameter when it is necessary to embed a custom form in your page template (such as a search field which posts to a different Web site). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle HTML DB must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV which passes a value back to a form element.
p_escape_html	Replacements for special characters that require an escaped equivalent. ■ <lt; for < ■ <gt; for > ■ <amp; for & This parameter is useful if you know your query will return illegal HTML.
p_max_elements	Limit on the number of rows that can be returned by your query. Limits the performance impact of user searches. By entering a value in this parameter, you force the user to search for a more narrow set of results.
p_attributes	Additional HTML attributes to use for the form item.
p_ok_to_query	Range of values is YES and NO. If YES, a popup returns first set of rows for the LOV. If NO, a search is initiated to return rows.

Example

The following example demonstrates how to generate a popup key select list from a shared list of values (LOV).

```
SELECT HTMLDB_ITEM.POPUPKEY_FROM_LOV (1,deptno,'DEPT') dt
FROM emp
```

POPUPKEY_FROM_QUERY Function

This function generates a popup key select list from a SQL query. Like other available functions in the HTMLDB_ITEM package, POPUPKEY_FROM_QUERY is designed to generate forms with F01 to F50 form array elements.

Syntax

```
HTMLDB_ITEM.POPUPKEY_FROM_QUERY(
    p_idx               IN    NUMBER,
    p_value             IN    VARCHAR2 DEFAULT,
    p_lov_query         IN    VARCHAR2,
    p_width             IN    VARCHAR2 DEFAULT,
    p_max_length        IN    VARCHAR2 DEFAULT,
    p_form_index        IN    VARCHAR2 DEFAULT,
    p_escape_html       IN    VARCHAR2 DEFAULT,
    p_max_elements      IN    VARCHAR2 DEFAULT,
    p_attributes        IN    VARCHAR2 DEFAULT,
    p_ok_to_query       IN    VARCHAR2 DEFAULT,
    p_item_id           IN    VARCHAR2 DEFAULT NULL,
    p_item_label        IN    VARCHAR2 DEFAULT NULL)
    RETURN VARCHAR2;
```

Parameters

Table 13–31 describes the some parameters in the POPUPKEY_FROM_QUERY function.

Table 13–31 POPUPKEY_FROM_QUERY Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, p_idx is a constant for a given column. Because of the behavior of POPUPKEY_FROM_QUERY, the next index value should be p_idx + 1. For example: `SELECT HTMLDB_ITEM.POPUPKEY_FROM_QUERY (1,deptno,'SELECT dname, deptno FROM dept') dt, HTMLDB_ITEM.HIDDEN(3,empno) eno`
p_value	Form element current value. This value should be one of the values in the P_LOV_QUERY parameter.
p_lov_query	LOV query used for this popup.
p_width	Width of the text box.
p_max_length	Maximum number of characters that can be entered in the text box.

Table 13–31 POPUPKEY_FROM_QUERY Parameters

Parameter	Description
p_form_index	HTML form on the page in which an item is contained. Defaults to 0 and rarely used. Only use this parameter when it is necessary to embed a custom form in your page template (such as a search field which posts to a different Web site). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle HTML DB must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV which passes a value back to a form element.
p_escape_html	Replacements for special characters that require an escaped equivalent. ■ <lt; for < ■ <gt; for > ■ <amp; for & This parameter is useful if you know your query will return illegal HTML.
p_max_elements	Limit on the number of rows that can be returned by your query. Limits the performance impact of user searches. By entering a value in this parameter, you force the user to search for a more narrow set of results.
p_attributes	Additional HTML attributes to use for the form item.
p_ok_to_query	Range of values is YES and NO. If YES, a popup returns first set of rows for the LOV. If NO, a search is initiated to return rows.
p_item_id	ID attribute of the form element.
p_item_label	Invisible label created for the item.

Example

The following example demonstrates how to generate a popup select list from a SQL query.

```
SELECT HTMLDB_ITEM.POPUPKEY_FROM_QUERY (1,deptno,'SELECT dname, deptno FROM
dept') dt
FROM emp
```

HTMLDB_APPLICATION

The HTMLDB_APPLICATION package is a PL/SQL package that implements the Oracle HTML DB rendering engine. You can use this package to take advantage of a number of global variables. Table 13–32 describes the global variables available in HTMLDB_APPLICATION.

Table 13–32 Global Variables Available in HTMLDB_APPLICATION

Global Variable	Description
G_USER	Specifies the currently logged in user.
G_FLOW_ID	Specifies the ID of the currently running application.
G_FLOW_STEP_ID	Specifies the ID of the currently running page.
G_FLOW_OWNER	Specifies the schema to parse for the currently running application.

Topics in this section include:

- Referencing Arrays
- Referencing Values Within an On Submit Process
- Converting an Array to a Single Value

Referencing Arrays

Items are typically HTML form elements such as text fields, select lists and check boxes. When you create a new form item using a wizard, the wizard uses a standard naming format. The naming format provides a handle so you can retrieve the value of the item later on.

If you need to create your own items, you can access them after a page is submitted by referencing HTMLDB_APPLICATION.G_F01 to HTMLDB_APPLICATION.G_F50 arrays. You can create your own HTML form fields by providing the input parameters using the format F01, F02, F03 and so on. You can create up to 50 input parameters ranging from F01 to F50. Consider the following example:

```
<INPUT TYPE="text" NAME="F01" SIZE="32" MAXLENGTH="32" VALUE="some value">

<TEXTAREA NAME="F02" ROWS=4 COLS=90 WRAP="VIRTUAL">this is the example of a text
area.</TEXTAREA>

<SELECT NAME="F03" SIZE="1">
```

```
<OPTION VALUE="abc">abc
<OPTION VALUE="123">123
</SELECT>
```

Since the F01 to F50 input items are declared as PL/SQL arrays, you can have multiple items named the same value. For example:

```
<INPUT TYPE="text" NAME="F01" SIZE="32" MAXLENGTH="32" VALUE="array element 1">
<INPUT TYPE="text" NAME="F01" SIZE="32" MAXLENGTH="32" ALUE="array element 2">
<INPUT TYPE="text" NAME="F01" SIZE="32" MAXLENGTH="32" VALUE="array element 3">
```

Note that following PL/SQL produces the same HTML as show in the previous example.

```
FOR i IN 1..3 LOOP
HTMLDB_ITEM.TEXT(P_IDX        => 1,
 p_value      =>'array element '||i ,
 p_size       =>32,
 p_maxlength  =>32);
END LOOP;
```

Referencing Values Within an On Submit Process

You can reference the values posted by an HTML form using the PL/SQL variable HTMLDB_APPLICATION.G_F01 to HTMLDB_APPLICATION.G_F50. Since this element is an array you can reference values directly. For example:

```
FOR i IN HTMLDB_APPLICATION.G_F01.COUNT LOOP
    htp.p('element '||I||' has a value of '||HTMLDB_APPLICATION.G_F01(i));
END LOOP;
```

Converting an Array to a Single Value

You can also use Oracle HTML DB public utility functions to convert an array into a single value. For example:

```
htp.p(HTMLDB_UTIL.TABLE_TO_STRING(HTMLDB_APPLICATION.G_F01));
```

This function is enables you to reference G_F01 to G_F50 values in an application process that performs actions on data. The following sample process demonstrates the insertion of values into an table:

```
FOR i IN 1..HTMLDB_APPLICATION.G_F01.COUNT LOOP
    INSERT INTO my_table (my_column) VALUES HTMLDB_APPLICATION.G_F01(i);
```

```
END LOOP;
```

HTMLDB_CUSTOM_AUTH

You can use HTMLDB_CUSTOM_AUTH to perform various operations related to authentication and session management.

Topics in this section include:

- APPLICATION_PAGE_ITEM_EXISTS Function
- CURRENT_PAGE_IS_PUBLIC Function
- DEFINE_USER_SESSION Procedure
- GET_NEXT_SESSION_ID Function
- GET_SECURITY_GROUP_ID Function
- GET_SESSION_ID Function
- GET_USER Function
- SESSION_ID_EXISTS Function
- SET_USER Procedure
- SET_SESSION_ID Procedure
- SET_SESSION_ID_TO_NEXT_VALUE Procedure

APPLICATION_PAGE_ITEM_EXISTS Function

This function checks for the existence of page level item within an application. This function requires the parameter p_item_name. This function returns a boolean value (true or false).

Syntax

```
FUNCTION APPLICATION_PAGE_ITEM_EXISTS(
    p_item_name    IN    VARCHAR2)
RETURN BOOLEAN;
```

CURRENT_PAGE_IS_PUBLIC Function

This function checks whether the current page's authentication attribute is set to **Page Is Public** and returns a boolean value (true or false)

> **See Also:** "Editing Page Attributes" on page 7-52 and "About Security" on page 7-55 for more information on setting this page attribute

Syntax

```
FUNCTION CURRENT_PAGE_IS_PUBLIC
RETURN BOOLEAN;
```

DEFINE_USER_SESSION Procedure

This procedure combines the `SET_USER` and `SET_SESSION_ID` functions to create one call.

Syntax

```
PROCEDURE DEFINE_USER_SESSION(
    p_user         IN    VARCHAR2)
    p_session_id   IN    NUMBER);
```

GET_NEXT_SESSION_ID Function

This function generates the next session ID from the Oracle HTML DB sequence generator. This function returns a number.

Syntax

```
FUNCTION GET_NEXT_SESSION_ID
RETURN NUMBER;
```

GET_SECURITY_GROUP_ID Function

This function returns a number with the value of the security group ID that identifies the workspace of the current user.

Syntax

```
FUNCTION GET_SECURITY_GROUP_ID
RETURN NUMBER;
```

GET_SESSION_ID Function

This function returns HTMLDB_APPLICATION.G_INSTANCE global variable. GET_SESSION_ID returns a number.

Syntax

```
PROCEDURE GET_SESSION_ID
RETURN NUMBER;
```

GET_USER Function

This function returns the HTMLDB_APPLICATION.G_USER global variable (VARCHAR2).

Syntax

```
FUNCTION GET_USER
RETURN VARCHAR2;
```

SESSION_ID_EXISTS Function

This function determines whether HTMLDB_APPLICATION.G_INSTANCE is set. SESSION_ID_EXISTS returns a BOOLEAN value (true or false).

Syntax

```
FUNCTION SESSION_ID_EXISTS
RETURN BOOLEAN;
```

SET_USER Procedure

This procedure sets the HTMLDB_APPLICATION.G_USER global variable. SET_USER requires the parameter P_USER (VARCHAR2) which defines a user ID.

Syntax

```
PROCEDURE SET_USER(
    p_user   IN    VARCHAR2)
```

SET_SESSION_ID Procedure

This procedure sets HTMLDB_APPLICATION.G_INSTANCE global variable. SET_SESSION_ID returns a number. This procedure requires the parameter P_SESSION_ID (NUMBER) which specifies a session ID.

Syntax

```
PROCEDURE SET_SESSION_ID(
    p_session_id    IN    NUMBER)
```

SET_SESSION_ID_TO_NEXT_VALUE Procedure

This procedure combines the operation of GET_NEXT_SESSION_ID and SET_SESSION_ID in one call.

Syntax

```
PROCEDURE SETsN_ID_TO_NEXT_VALUE;
```

Part III

Administration

Part III describes all tasks performed by an Oracle HTML DB administrator. An Oracle HTML DB administrator manages an entire Oracle HTML DB development environment instance through the Oracle HTML DB Administration Services application. Common Oracle HTML DB administrator tasks include creating and managing workspaces, translating an application, and managing activities, log files, and sessions.

Part III contains the following chapters:

14

Administering Workspaces

This section describes tasks an Oracle HTML DB administrator performs when administering workspaces.

This section contains the following topics:

About the Oracle HTML DB Administrator

In the Oracle HTML DB development environment, users log in to a shared work area called a workspace. Users are divided into three primary roles:

- Developer
- Workspace administrator
- Oracle HTML DB administrator

A developer can create and edit applications. A Workspace administrator performs administrator tasks specific to their workspace such as managing user accounts, monitoring workspace activity, and viewing log files. An Oracle HTML DB administrator manages an entire Oracle HTML DB development environment instance. To perform these tasks, an Oracle HTML DB administrator logs into the

Oracle HTML DB Administration Services application. Your Oracle HTML DB Administration Services application can be found at the following location:

```
http://server:port/pls/Database Authentication Descriptor/htmldb_admin
```

> **See Also:** *Oracle HTML DB Quick Installation Guide* for your platform for more information on installing Oracle HTML DB

Viewing Workspace Reports

Oracle HTML DB administrators can view detailed information about a specific workspace by viewing the Workspace Utilization Report.

To view a workspace report:

1. Log in to Oracle HTML DB Administration Services.
2. Under Manage Workspaces, select **Report Workspace Attributes**.
3. Select a workspace from the Workspace list and click **Go**.

   The Workspace Utilization Report appears. Table 14–1 describes the various sections of the Workspace Utilization Report.

Table 14–1 Workspace Utilization Report

Report	Description
Workspace Information	Enables administrators to edit workspace information.
Workspace Schemas and Default Tablespaces	Enables administrators to manage workspace to schema mappings. **See Also:** "Managing the Schemas Associated with a Workspace" on page 14-8
Workspace Schemas Utilizing Space in Tablespaces	Displays a report that details tablespace utilization.
Applications	Displays a report that lists all applications within the current workspace.
Application Developers	Displays a report that lists all application developers within the current workspace.
Cookie Users	Enables administrators to manage user accounts. **See Also:** "Managing Users in a Workspace" on page 14-6

Table 14–1 Workspace Utilization Report

Report	Description
Objects Used By Workspace	Displays a report that lists objects used in the current workspace.
Service Change Requests	Enables administrators to manage change requests for the current workspace, or to view a report of all change requests in an Oracle HTML DB development instance. **See Also:** "Managing a Service and Change Request" on page 14-4
Developer Activity	Displays a report that details developer activity by date.

Creating a Workspace

When a user logs into the Oracle HTML DB they log in to a shared work area called a workspace. Each workspace is an area within the Oracle HTML DB development environment where multiple developers can create applications. Each workspace has a unique ID and name. In order to make changes to their workspace, Workspace administrators submit change request to an Oracle HTML DB administrator. Only an Oracle HTML DB administrator can create a new workspace.

Topics in this section include:

- Specifying a Provisioning Mode
- Managing a Service and Change Request
- Creating a Workspace Without a Request

See Also: "Managing Development Services" on page 11-8

Specifying a Provisioning Mode

As an Oracle HTML DB administrator, you determine how the process of creating (or provisioning) a workspace works for your Oracle HTML DB development environment.

In **manual** provision mode, an Oracle HTML DB administrator creates new workspaces and notifies the Workspace administrator of the login information. In **request** provision mode, users request workspaces directly using an automated process. In this scenario, users use a link on the login page to access a request form. Once the workspace request has been granted, they are e-mailed the appropriate login information.

To specify a provisioning mode:

1. Log in to Oracle HTML DB Administration Services.
2. Under Manage HTML DB Service, select **Toggle Provisioning Status**.
3. Select one of the following:
   - **Manual**
   - **Request**
4. Click **Apply Changes**.

> **Note:** Before users can request a workspace or change their passwords, an Oracle HTML DB administrator must configure engine settings. For more information, see "Managing Engine Settings" on page 15-5.

Managing a Service and Change Request

Oracle HTML DB administrators can make modifications to a workspace (such as such as adding an additional schema or increasing the disk space limit) by approving a change request from a Workspace administrator.

Viewing a Pending Service or Change Request

You can view a summary of pending service requests and change requests in the Notifications list on the Administration home page.

To view pending service and change requests:

1. Log in to Oracle HTML DB Administration Services.
2. Locate the Notifications list on the lower right side of the page (See Figure 14–1).

Figure 14–1 Notifications List

The Notifications list displays a summary of total and pending service and change requests.

3. To view additional details, click the appropriate service request or change request number.

   The appropriate Change Request page appears.

To view pending requests from the Workspace Administration tab:

1. Log in to Oracle HTML DB Administration Services.
2. Select the **Workspace Administration** tab.
3. Locate a workspace as follows:
   - To locate a specific workspace, type the workspace name in the Search field and click **Go**.
   - To view all workspaces, leave the Search field blank and click **Go**.
4. Click the view icon to the left of the appropriate workspace name.

   The Workspace Utilization Report appears.
5. Under Available Reports, click **Service Change Requests**.
6. Select a specific request, or click **View All Change Requests**.

Approving a Service or Change Request

To approve a pending service request:

1. Navigate to the Service Change Request page as described in "Viewing a Pending Service or Change Request" on page 14-4.
2. Click **Provision**.

   The Provisioning Administration page appears.
3. Select one of the following:
   - To approve the request, click **Approve**.
   - To decline the request, click **Decline**.
4. Follow the on-screen instructions.

To approve a pending change request:

1. Navigate to the Service Change Request page as described in "Viewing a Pending Service or Change Request" on page 14-4.

2. Click **View Request**.

   The Process Change Request page appears.
3. Review the displayed reports.
4. Select one of the following:
   - To approve a request for a schema, click **Create Schema**.
   - To approve a request for additional disk space, click **Provision Space**.
   - To approve a request to terminate the service, click **Terminate Service**
5. To deny a request, click **Deny Request**.
6. Follow the on-screen instructions.

Creating a Workspace Without a Request

Administrators can create a workspace manually by running the Provision Workspace Wizard. You can access this wizard from either the Oracle HTML DB Administration Services Home page or the Workspace Administration tab.

To create a workspace manually:

1. Log in to Oracle HTML DB Administration Services.
2. Under Manage Workspaces, select **Create New Workspace**.

   The Provision Workspace Wizard appears.
3. Specify a workspace name, ID, and description and click **Next**.
4. Select a schema, or enter the name for a new schema, followed by a password, and initial disk space quota.
5. Click **Next**.
6. Specify a Workspace administrator by providing a username, password, and e-mail address and click **Next**.
7. Confirm your selections and click **Provision**.

Managing Users in a Workspace

Oracle HTML DB administrators can manage all user accounts within an Oracle HTML DB instance on the Manage Application Developers and Users page. User

accounts are particularly useful if a workspace utilizes internal "Cookie User" authentication.

See Also:

- "Managing Your Development Workspace" on page 11-1
- "About HTML DB Account Credentials" on page 10-14 for more information on implementing internal Cookie User (or HTML DB Account Credentials) authentication.

To create a new user account:

1. Log in to Oracle HTML DB Administration Services.
2. Under Manage Workspaces, select **Manage Application Developers**.

   The Manage Application Developers and Users page appears.
3. Click **Create**.
4. Under User Attributes, enter the appropriate information. Fields marked with a red asterisk (*) are required.
5. Under Password, type a case sensitive password for this account.
6. Under Developer Privileges, specify the developer's privileges.

   Developers having the **Admin** privilege have access to the Administration Services page and all the functionality described in "Managing Your Development Workspace" on page 11-1.
7. Click **Create** or **Create and Create Another**.

To edit an existing user account:

1. Log in to Oracle HTML DB Administration Services.
2. Under Manage Workspaces, select **Manage Application Developers**.

   The Manage Application Developers and Users page appears.
3. Locate a user as follows:
   - To locate a specific user, type a username in the Search field and click **Go**.
   - To view all users, leave the Search field blank and click **Go**
4. Click the edit icon adjacent to appropriate username.
5. Follow the on-screen instructions.

Managing the Schemas Associated with a Workspace

A workspace can have multiple associated schemas. By associating a workspace with a schema, you can:

- Build applications that interact with the database objects in that schema.
- Create new database objects in that schema.

To view the schema associated with a workspace:

1. Log in to Oracle HTML DB Administration Services.
2. Under Manage Workspaces, select **Manage Schema to Workspace Assignments**.

   The Schemas Provisioned by Workspace page appears.
3. To view information about an existing schema, click the edit icon.
4. To create a new schema association, click **Create**.
5. Follow the on-screen instructions.

Removing a Workspace

Removing a workspace does not remove the associated schema or schemas. To remove the associated schemas, a database administrator (DBA) must use a standard database administration tool such as Oracle Enterprise Manager or SQL*Plus.

See Also:

- *Oracle Enterprise Manager Administrator's Guide*
- *SQL*Plus User's Guide and Reference*

To remove a workspace:

1. Log in to Oracle HTML DB Administration Services.
2. Under Manage Workspaces, select **Remove Workspace**.
3. Select a workspace name and click **Next**.
4. Follow the on-screen instructions.

Exporting and Importing a Workspace

To move a workspace and all associated users to a new Oracle HTML DB development instance, you must export the workspace. When you export a workspace, Oracle HTML DB generates a text file. This file contains information about your workspace, all the users in your workspace, and any groups in your workspace (if applicable). You can use this file to import your workspace into another Oracle HTML DB instance.

Keep in mind, this method only imports workspace, users, and groups. This file does not contain:

- The schemas associated with this workspace, or the objects in those schemas.
- Any applications, images, cascading style sheets and static text files.

All of these items must be exported separately.

See Also:

- "Exporting and Importing Applications" on page 10-2
- "About Managing Database Objects" on page 10-3
- "Uploading CSS, Images, and Static Files" on page 10-8

To export a workspace:

1. Log in to Oracle HTML DB Administration Services.
2. Under Manage Workspaces, select **Export Workspace**.
3. Select a workspace name and click **Export**.
4. To export the selected workspace, click **Save File**.
5. Follow the on-screen instructions.

To import a workspace:

1. Log in to Oracle HTML DB Administration Services.
2. Under Manage Workspaces, select **Import Workspace**.
3. Select a workspace name and click **Next**.
4. To install the workspace, click **Install**.
5. Follow the on-screen instructions.

15

Managing Services

This section provides information about managing Oracle HTML DB services. Oracle HTML DB administrators can use the Manage HTML DB Services application to manage log files, purge session state, monitor developer activities, and manage engine settings.

This section contains the following topics:

- Managing Logs
- Managing Session State
- Monitoring Activities
- Managing Engine Settings

> **See Also:** "About the Oracle HTML DB Administrator" on page 14-1 for more information on different administrator roles

Managing Logs

Oracle HTML DB administrators can manage the following log files on the Manage Logs and Files page:

- Developer activity logs
- External click counting log
- SQL Workshop logs
- Page View Activity logs

To manage log files:

1. Log in to Oracle HTML DB Administration Services. (See "About the Oracle HTML DB Administrator" on page 14-1.)

2. Under Manage HTML DB Service, select **Manage Logs**.

   The Manage Logs page appears.

3. Select one of the following:
   - **Developer activity logs, review with option to delete entries**
   - **External click counting log, review with option to truncate**
   - **Review SQL Workshop logs**
   - **Review page view activity log, with option to truncate**

Deleting Developer Activity Log Entries

Clicking **Developer activity logs, review with option to delete entries** links you to the Developer Activity Log.

To delete a specific number of log entries:

1. Click **Manage**.
2. Specify the age of the entries to be deleted and click **Delete Entries**.

Deleting Click Counting Log Entries

Clicking **External click counting log, review with option to truncate** links you to the Click Counting Log.

To delete a specified number of log entries:

1. Click **Manage**.
2. Specify the age of the entries to be deleted and click **Delete Entries**.

Deleting SQL Workshop Logs

Clicking **Review SQL Workshop logs** links you to SQL Workshop logs. These logs maintain a history of recent commands and scripts run in the SQL Command Processor.

To delete or truncate log files entries:

1. Select one of the following:
   - Script File executions log entries
   - Control File execution log entries

- SQL Command Processor history log entries
- SQL Archives entries

2. To delete entries by age:
   - Specify the age of the entries to deleted
   - Click Delete Entries
3. To delete all entries, click **Truncate Log**.

See Also: "Accessing the SQL Command History" on page 5-12

Deleting User Activity Log Entries

Clicking **Review page view activity log, with option to truncate** links you to the Manage Activity Logs page. Activity logs track user activity for an application. Developers enable logging within their application on the Edit Application Attributes page.

See Also: "About Application Definition" on page 7-30 for more information on enabling logging on an application

The HTML DB engine actually uses two logs to track user activity. At any given time, one log is designated as current. For each rendered page view, the HTML DB engine inserts one row into the log file. A log switch occurs at the interval listed on the Manage Activity Logs page. At that point, the HTML DB engine removes all entries in the noncurrent log and designates it as current.

To truncate the activity logs manually:

1. Click **Truncate Logs**.
2. Click either **Truncate Log 1** or **Truncate Log 2**.

Managing Session State

A session is a logical construct that establishes persistence (or stateful behavior) across page views. Each session is assigned a unique ID which the HTML DB engine uses to store and retrieve an application's working set of data (or session state) before and after each page view. Sessions persist in the database until purged by an administrator.

An Oracle HTML DB administrator can view session state statistics and purge session state on the Session State Management page.

Topics in this section include:

- Purging Sessions by Age
- Viewing Session Details Before Purging
- Viewing Session Statistics Before Purging

See Also: "Understanding Session State Management" on page 6-12

Purging Sessions by Age

Using the Purge Session page, administrators can purge sessions by age.

To view specific session details:

1. Log in to Oracle HTML DB Administration Services.
2. Select the **Service Administration** tab.
3. Select **Purge old sessions by age**.
4. On the Purge Session page, specify:
   - The maximum number of sessions to be purged
   - The age of session to be purged
5. To view a report of session statistics, click **Count Sessions**.
6. To purge the selected sessions, click **Purge Sessions**.

Viewing Session Details Before Purging

Before purging sessions, administrators can use the Recent Sessions page to first view a listing of recent sessions and then drill down on session details.

To purge sessions by age:

1. Log in to Oracle HTML DB Administration Services.
2. Select the **Service Administration** tab.
3. Select **Report recent sessions with drill down to session details**.
4. On the Recent Sessions page, you can:
   - Click a session ID to view additional details.
   - Click Purge Session to delete the displayed sessions.

Viewing Session Statistics Before Purging

On the Session State Statistics page, administrators can view statistics about current sessions prior to purging.

To view session state statistics:

1. Log in to Oracle HTML DB Administration Services.
2. Select the **Service Administration** tab.
3. Select **Report session counts**.
4. Click **Purge Sessions** to delete the current sessions.

Monitoring Activities

Oracle HTML DB administrators can monitor user activity by accessing a number of charts and reports on the Monitoring page.

To monitor user activity:

1. Log in to Oracle HTML DB Administration Services.
2. Select the **Monitoring** tab.

   The Monitoring page appears.
3. Select a chart or report to review.

Managing Engine Settings

HTML DB engine settings are named substitution value pairs defined by an Oracle HTML DB administrator. Engine setting are used internally by the HTML DB engine to determine a provisioning mode and to configure the HTML DB engine to send mail.

`SERVICE_REQUEST_FLOW` determines whether Oracle HTML DB is in manual provision mode or request provision mode. When in request provision mode, users can use a link on the login page to request a workspace.

When an administrator logs into Oracle HTML DB Administration Services, they can create and delete this engine setting by selecting Toggle Provisioning Mode. Clicking this link and selecting **Request** creates a preference named `SERVICE_REQUEST_FLOW` that has a value of 4700. Reversely, selecting **Manual** removes this engine setting.

If you enable request provision mode or enable users to reset their passwords using a link on the login page, your must configure Oracle HTML DB to send mail. In order to enable Oracle HTML DB to send mail, you must configure the following engine settings.

- `DEVELOPMENT_SERVICE_URL` - If you are running in request provisioning mode, the value of this setting is used in the e-mail when the request is approved. This setting defines the URL for the service. If this setting is not present, the URL will be derived from your environment.
- `SMTP_HOST_ADDRESS` - Defines the server address of the SMTP server. On installation, this will be set to `localhost`. If you are using another server for SMTP relaying, change `localhost` to that server's address.
- `SMTP_HOST_PORT` - Defines the port the SMTP server listens to for mail requests. By default, this setting will be set to 25 at the time of installation.
- `SMTP_FROM` - Defines the "from" address when an administrative tasks such as approving a provision request, or resetting a password generates an e-mail.

16

Managing Globalization

This section describes how to translate an application built in Oracle HTML DB.

This section contains the following topics:

About Translating an Application and Globalization Support

In Oracle HTML DB you can develop applications that can run concurrently in different languages. A single Oracle database instance and Oracle HTML DB can support multiple database sessions customized to support different language.

In general, translating an Oracle HTML DB application involves the following steps:

- Map primary and target application IDs
- Seed and export text to a file for translation
- Translate the text in the file
- Apply and publish the translated file

See Also: "Understanding the Translation Process" on page 16-6

About Language Identification

After you create an application, you specify a language preference on the Edit Application Attributes page. Under Globalization, you select a primary application language and select how the HTML DB engine determines the application language. You can specify to have the application language based on the user's browser language preference, an application preference, or an item preference.

> **See Also:** "Specifying the Primary Language for an Application" on page 16-4

How Translated Applications Are Rendered

Once Oracle HTML DB determines the language for an application, the HTML DB engine alters the database language for a specific page request. It then looks for a translated application in the appropriate language. If the HTML DB engine finds that language, it render the application using that definition. Otherwise, it renders the application in the base (or primary) application language.

Note that the text that displays within an application is not translated on the fly. Oracle HTML DB dynamically collects page attributes from either a base language application definition or an alternative application definition.

> **See Also:** "About Dynamic Translation Text Strings" on page 16-3 and "Translating Data that Supports List of Values" on page 16-14

About Translatable Components

When you build an application in Oracle HTML DB, you define a large number of declarative attributes such as field labels, region headings, page header text, and so on. Using the steps described in this chapter, you can make all the application definition attributes within your application translatable.

About Messages

If your application includes PL/SQL regions or PL/SQL processes, you may need to translate any generated HTML or text. Within Oracle HTML DB this type of generated HTML and text are called "messages." You can define all messages and translate them on the Translatable Messages page. You can use the `HTMLDB_LANG.MESSAGE` API to translate text strings from PL/SQL stored procedures, functions, triggers, packaged procedures and functions.

See Also: "Translating Messages Used in PL/SQL Procedures" on page 16-12

About Dynamic Translation Text Strings

Dynamic translations are used for database data that needs to be translated at runtime. For example, you might use a dynamic translation to translate a list of values based on a database query. A dynamic translation consists of a "translate-from" language string, a language code, and a "translate-to" string. You can also use the `HTMLDB_LANG.LANG` API to retrieve dynamic translations programmatically.

See Also: "Translating Data that Supports List of Values" on page 16-14

About Translating Templates

By default, templates in Oracle HTML DB are not translatable and therefore not included in the generated translation file. Generally, templates do not and should not contain translatable text. However, if you need to mark a template as translatable, mark the Translatable check box on the Edit Page Template page.

To identify a template as translatable:

1. Click the **Build** icon.
2. Select the **Templates** tab.
3. Locate the template you wish to edit and click the edit icon.
4. Under Template Identification, select **Translatable**.

One way to include translatable text at the application level is to define the translatable text using static substitution strings. Since application level attributes are translated any text defined as a static substitution strings will be included in the generated translation file.

See Also:

- "Editing Templates" on page 7-17
- "About Static Substitution Strings" on page 7-36

Specifying the Primary Language for an Application

Globalization attributes specify how the HTML DB engine determines the primary language of an application.

To edit Globalization attributes:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Edit Attributes** icon.

   The Edit Application Attributes page appears.
4. Scroll down to Globalization.
5. From **Application Primary Language**, select the language in which the application is being developed.
6. From **Application Language Derived From**, specify how the HTML DB engine determines (or derives) the application language. Available options are described in Table 16–1.

Table 16–1 Application Language Derived From Options

Option	Description
No NLS (Application not translated)	Select this option if the application will not be translated.
Use Application Primary Language	Determines the application primary language based on the Application Primary Language attribute (see step 5).
Browser (use browser language preference)	Determines the application primary language based on the user's browser language preference.
Application Preference (use FSP_LANGUAGE_PREFERENCE)	Determines the application primary language based a value defined using the `HTMLDB_UTIL.SET_PREFERENCE` API. Select this option to maintain the selected language preference across multiple log ins. **See Also:** "SET_PREFERENCE Procedure" on page 13-9
Item Preference (use item containing preference)	Determines based on an application level item called `FSP_LANGUAGE_PREFERENCE`. Using this option requires Oracle HTML DB to determine the appropriate language preference every time the user logs in.

See Also:

- "Editing Application Attributes" on page 7-29
- "About Globalization" on page 7-35
- "About Oracle HTML DB Globalization Codes" on page 16-16

Using Format Masks for Items

The HTML DB engine applies Globalization settings for each rendered page. This default behavior can impact the display of certain items such as numbers and dates.

For example, suppose your application determines the application language based on the user's browser language preference. If the HTML DB engine determines the users's browser language preference is French, it displays dates and numbers in a format that conforms to French standards. You can override this default behavior and explicitly control how items display by applying a format mask. You apply a format mask by making a selection from the Display As list:

- When you create the item
- After you create the item by editing the item attributes

To edit item attributes

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Navigate to the appropriate page.
4. Under Items, drill down on the item name.

   The Edit Page Item page appears.
5. Under Identification, make a select from the Display As list.

   See Also: "About Items" on page 7-43 for more information on item attributes.

Translating Applications for Multibyte Languages

If your application needs to run in several languages (such as Chinese or Japanese) simultaneously, you should consider configuring your database with a character set to support all of the languages. The same character set has to be configured in the corresponding DAD (Data Access Description) in mod_plsql. UTF8 and AL32UTF8 are the character sets you can use to support almost all languages around the world.

See Also: *Oracle Database Globalization Support Guide*

Understanding the Translation Process

To translate an application developed in Oracle HTML DB, you must map the primary and target application IDs, seed and export text to a translation file, translate the text, and then apply and publish the translation file.

Topics in this section include:

- Navigating to the Translate Application Page
- Mapping Primary and Target Application IDs
- Seeding and Exporting Text to a Translation File
- Translating the XLIFF File
- Uploading and Publishing a Translated XLIFF Document

Navigating to the Translate Application Page

You perform the translation process on the Translate Application page.

To navigate to the Translate Application page:

1. Click the **Build** icon.
2. From the Available Applications list, select an application and click **Go**.
3. Select the **Utilities** tab.
4. Click **Translate Application**.

   The Translate Application page appears.

Mapping Primary and Target Application IDs

The first step in translating an application is to map the primary and target application IDs. The primary application is the application to be translated. The target application is the resulting translated application.

To map the primary and target application IDs:

1. Navigate to the Translate Application page. (See "Navigating to the Translate Application Page".)

2. On the Translate Application page, select **Map your primary language application to a translated application ID**.

   The Application Mappings page appears.

3. Click **Create**.

4. On the Translation Application Mapping page:

   - In Translation Application, type a numeric application ID to identify the target application.
   - From Translation Application Language Code, select the language you are translating to.
   - In Image Directory, enter the directory where images will be obtained.

     This attribute determines the virtual directory for translated images. For example, if your primary language application had an image prefix of `'/images/'`, you could define additional virtual directories for other languages such as `'/images/de/'` for German or `'/images/es/'` for Spanish.

5. Click **Create**.

Seeding and Exporting Text to a Translation File

The second step in translating an application is to seed the translation table and then export the translation text to a translation file.

Seeding Translatable Text

Seeding translatable text copies all strings that may require translation to an Oracle HTML DB database table that contains that contains the original language version and the possibility of storing the translated version.

To seed translatable text:

1. Navigate to the Translate Application page. (See "Navigating to the Translate Application Page" on page 16-6.)

2. On the Translate Application page, select **Seed and export the translation text of your application into a translation file**.

3. From Language Mapping, select the appropriate primary and target application ID map.

4. Click **Seed Translatable Text**.

The XLIFF Export page appears.

Note: XML Localization Interchange File Format (XLIFF) is a XML-based format for exchanging localization data. For more information on the XLIFF, or to view the XLIFF specification see:

`http://www.xliff.org`

Exporting Text to a Translation File

Once you have seeded translatable text, a status box displays at the top of the XLIFF Export page indicating the total number of attributes that may require translation as well as the number of:

- Existing updated attributes that may require translation
- New attributes that may require translation
- Purged attributes that no longer require translation

You can use this information to determine whether you need to export translatable text for an entire application or just a specific page.

The XLIFF Export page is divided into two sections. Use the upper half of the page to export translatable text for an entire application (that is, all pages, lists of values, messages, and so on). Use the lower section to export translatable text for a specific page.

To export translatable text for an entire application:

1. Seed the translatable text as described in the previous procedure, "Seeding Translatable Text" on page 16-7.
2. Under **Step 2, Export XLIFF**:
   - From Application, select the appropriate primary and target application ID map
   - Specify whether to include XLIFF target elements
   - Under Export, specify what translation text is included in your XLIFF file
   - Click **Export XLIFF for Application**
3. Follow the on-screen instructions.

To export translatable text for a specific page:

1. Seed translatable text as described in "Seeding Translatable Text" on page 16-7.

2. Under **Export XLIFF for specific Page**:
   - From Application, select the appropriate primary and target application ID map
   - Specify whether to include XLIFF target elements
   - Under Export, specify what translation text is included in your XLIFF file
   - Click **Export XLIFF for Page**
3. Follow the on-screen instructions.

About Include XLIFF Target Elements When Oracle HTML DB generates an XLIFF document, each document contains multiple translation units. Each translation unit consists of a source element and a target element. If you have not previously translated an application, you must include source and target elements. However, if you have a previous translation, you have the option of disabling this option and only generating a file containing source elements.

About Export Use **Export** to specify what translation text is included in your XLIFF file. Select **All translatable elements** to include all translation text for an application. In contract, select **Only those elements requiring translation** to include only new elements that have not yet been translated.

Translating the XLIFF File

After you export a translatable file to XLIFF format, you can translate it into the appropriate languages. Since XLIFF is an open standard XML file for exchanging translation, most translation vendors should support it. Oracle HTML DB only supports XLIFF files encoded in UTF-8 character sets. In other words, it exports XLIFF files for translation in UTF-8 and assumes that the translated XLIFF files will be in the same character set.

Translation is a time consuming task. Oracle HTML DB supports incremental translation so that application development can be done in parallel with the translation. An XLIFF file can be translated and uploaded to Oracle HTML DB even when only part of the XLIFF file is translated. For strings that have no translation in the corresponding translated application, Oracle HTML DB uses the corresponding ones in the primary language.

See Also: For more information on the XLIFF, or to view the XLIFF specification see:

`http://www.xliff.org`

Uploading and Publishing a Translated XLIFF Document

Once your XLIFF document has been translated, the next step is to upload it back into Oracle HTML DB.

To upload a translated XLIFF document:

1. Navigate to the Translate Application page. (See "Navigating to the Translate Application Page".)
2. On the Translate Application page, select **Apply your translation file and publish**.
3. Click **Upload XLIFF**.
4. On the XLIFF Upload page:
   - Specify a title
   - Enter a description
   - Click **Browse** and locate the file to be uploaded
   - Click **Upload XLIFF File**

   The uploaded document appears in the XLIFF Files repository.

Once you upload an XLIFF document, the next step is to apply the XLIFF document and then publish the translated application. When you apply an XLIFF document, the HTML DB engine parses the file and then updates the translation tables with the new translatable text.

Publishing your application creates a copy of the base language application, substituting the translated text strings from your translations table. This published application can then be used to render your application in alternate languages.

Remember that in order to run an application in an alternative language, you need to run it with Globalization settings that will cause an alternative language version to display. For example, if the language is derived from the browser language, you must set the browser language to the same language as the translated application.

> **See Also:** "Specifying the Primary Language for an Application" on page 16-4

To apply and publish a translated XLIFF document:

1. Navigate to the Translate Application page. (See "Navigating to the Translate Application Page".)
2. On the Translate Application page, select **Apply your translation file and publish**.
3. In the XLIFF Files repository, click the view icon.
4. From Apply to, select the appropriate primary and target application ID map.
5. Click **Apply XLIFF Translation File**.
6. Click **Publish Application**.

To delete an uploaded XLIFF document:

1. Navigate to the Translate Application page. (See "Navigating to the Translate Application Page".)
2. On the Translate Application page, select **Apply your translation file and publish**.
3. In the XLIFF Files repository, select the check box to the left of the document title.
4. Click **Delete Checked**.

You should verify the existence of the translated application once it is published. Translated applications do not display in the Available Applications list on the Application Builder home page. Instead, use the Application Navigation pane on the left side of the page.

Note that in order for a translated application to appear in Application Builder, you need to make sure the you have correctly configured the application Globalization attributes.

> **Note:** "Specifying the Primary Language for an Application" on page 16-4

Translating Messages Used in PL/SQL Procedures

If your application includes PL/SQL regions or PL/SQL processes or calls PL/SQL package, procedures, or functions, you may need to translate generated HTML. First, you define each message on the Translatable Messages page. Second, you use the `HTMLDB_LANG.MESSAGE` API to translate the messages from PL/SQL stored procedures, functions, triggers, or packaged procedures and functions.

Defining Translatable Messages

You create translatable messages on the Translate Messages page.

To define a new translation message:

1. Navigate to the Translate Application page. (See "Navigating to the Translate Application Page".)
2. On the Translate Application page, select **Optionally translate messages which are used by PL/SQL procedures and functions**.
3. On the Translate Messages page, click **Create**.
4. On the Identify Text Message page:
   - In Name, type a name to identify the text message
   - In Language, select the language for which the message would be used
   - In text, type the text to be returned when the text message is called.

     For example, you could define the message `GREETING_MSG` in English as:

     ```
     Good morning %0
     ```

     Or, you could define the message `GREETING_MSG` in German as:

     ```
     Guten Tag %0
     ```
5. Click **Create**.

HTMLDB_LANG.MESSAGE API

Use the `HTMLDB_LANG.MESSAGE` API to translate text strings (or messages) generated from PL/SQL stored procedures, functions, triggers, packaged procedures and functions.

Syntax

```
HTMLDB_LANG.MESSAGE (
    p_name     IN    VARCHAR2 DEFAULT NULL,
    p0         IN    VARCHAR2 DEFAULT NULL,
    p1         IN    VARCHAR2 DEFAULT NULL,
    p2         IN    VARCHAR2 DEFAULT NULL,
    ...
    p9         IN    VARCHAR2 DEFAULT NULL,
    p_lang     IN    VARCHAR2 DEFAULT NULL)
    RETURN VARCHAR2;
```

Parameters

Table 16–2 describes the parameters available in the HTMLDB_LANG.MESSAGE.

Table 16–2 HTMLDB_LANG.MESSAGE Parameters

Parameter	Description
p_name	Name of the message as defined in Oracle HTML DB.
p0 ... p9	Dynamic substitution value. p0 corresponds to 0% in the message. p1 corresponds to 1% in the message. p2 corresponds to2% in the message and so on.
p_lang	Language code for the message to be retrieved. If not specified, Oracle HTML DB uses the current language for the user as defined in the Application Language Derived From attribute. **See Also:** "Specifying the Primary Language for an Application" on page 16-4

Example

The following example assumes you have defined a message called GREETING_MSG in your application in English as Good morning%0 and in German as Guten Tag%1. The following example demonstrates how you could invoke this message from PL/SQL:

```
BEGIN
    --
    -- Print the greeting
    --
    HTMLDB_LANG.MESSAGE('GREETING_MSG', v('APP_USER'));
END;
```

How p_lang attribute is defined depends on how the HTML DB engine derives the Application Primary Language. For example, if you are running the application in German and the previous call is made HTMLDB_LANG.MESSAGE, the HTML DB engine first looks for a message called GREETING_MSG with a LANG_CODE of de. If it does not find anything, then it will revert to the Application Primary Language attribute. If it still does not find anything, the HTML DB engine looks for a message by this name with a language code of en-us.

> **See Also:** "Specifying the Primary Language for an Application" on page 16-4 for more information on the Application Primary Language attribute.

Translating Data that Supports List of Values

You create a dynamic translation to translate dynamic pieces of data. For example, you might use a dynamic translation on a list of values based on a database query.

Dynamic translations differ from messages in that you query a specific string rather then a message name. You define dynamic translations on the Dynamic Translations page. You then use the HTMLDB_LANG.LANG API to return the dynamic translation string identified by the parameter p_primary_text_string.

Defining a Dynamic Translation

You define dynamic translations on the Dynamic Translations page. A dynamic translation consists of a "translate-from" language string, a language code, and a "translate-to" string.

To define a dynamic translation:

1. Navigate to the Translate Application page. (See "Navigating to the Translate Application Page".)
2. On the Translate Application page, select **Optionally identify any data that needs to be dynamically translated to support SQL based lists of values**.
3. On the Dynamic Translations page, click **Create** and specify the following:
   - In Language, select a target language
   - In Translate From Text, type the source text to be translated
   - In Translate To, type the translated text
4. Click **Create**.

HTMLDB_LANG.LANG API

Syntax

```
HTMLDB_LANG.LANG (
    p_primary_text_string     IN    VARCHAR2 DEFAULT NULL,
    p0                        IN    VARCHAR2 DEFAULT NULL,
    p1                        IN    VARCHAR2 DEFAULT NULL,
    p2                        IN    VARCHAR2 DEFAULT NULL,
    ...
    p9                        IN    VARCHAR2 DEFAULT NULL,
    p_primary_language        IN    VARCHAR2 DEFAULT NULL)
    RETURN VARCHAR2;
```

Parameters

Table 16–3 describes the parameters available in the HTMLDB_LANG.LANG.

Table 16–3 HTMLDB_LANG.LANG Parameters

Parameter	Description
p_primary_string	Text string of the primary language. This will be the value of the Translate From Text in the dynamic translation.
p0 ... p9	Dynamic substitution value. p0 corresponds to 0% in the in the translation string. p1 corresponds to 1% in the in the translation string. p2 corresponds to 2% in the in the translation string and so on.
p_primary_language	Language code for the message to be retrieved. If not specified, Oracle HTML DB uses the current language for the user as defined in the Application Language Derived From attribute. **See Also:** "Specifying the Primary Language for an Application" on page 16-4

Example

Suppose you have a table that defines all primary colors. You could define a dynamic message for each color and then apply the LANG function to the defined values in a query. For example:

```
SELECT HTMLDB_LANG.LANG(color)
  FROM my_colors
```

For example, suppose you were running the application in German and`RED` was a value for the color column in `my_colors` table. If you defined the German word for red, the previous example would return `ROT`.

About Oracle HTML DB Globalization Codes

If you are building a multilingual application, it is important to understand how Oracle HTML DB globalization codes impact the way in which your application runs. These codes are set automatically based on the application-level Globalization attributes you select.

See Also: "Specifying the Primary Language for an Application" on page 16-4

`NLS_LANGUAGE` and `NLS_TERRITORY` determine the default presentation of number, dates, and currency.

Table 16–4 describes the globalization codes in Oracle HTML DB.

Table 16–4 Oracle HTML DB Globalization Codes

Language Name	Language Code	NLS_LANGUAGE	NLS_TERRITORY
Arabic	ar	ARABIC	
Assamese	as	ASSAMESE	INDIA
Bengali	bn	BANGLA	
Bulgarian	bg	BULGARIAN	BULGARIA
Catalan	ca	CATALAN	CATALONIA
Chinese (China)	zh-cn	SIMPLIFIED CHINESE	CHINA
Chinese (Hong Kong SAR)	zh-hk	TRADITIONAL CHINESE	HONG KONG
Chinese (Singapore)	zh-sg	SIMPLIFIED CHINESE	SINGAPORE
Chinese (Taiwan)	zh-tw	TRADITIONAL CHINESE	TAIWAN
Chinese	zh	SIMPLIFIED CHINESE	CHINA
Croatian	hr	CROATIAN	CROATIA
Czech	cs	CZECH	CZECH REPUBLIC
Danish	da	DANISH	DENMARK

Table 16–4 Oracle HTML DB Globalization Codes

Language Name	Language Code	NLS_LANGUAGE	NLS_TERRITORY
Dutch (Netherlands)	nl	DUTCH	THE NETHERLANDS
English (United States)	en-us	AMERICAN	AMERICA
English	en	ENGLISH	
Estonian	et	ESTONIAN	ESTONIA
Finnish	fi	FINNISH	FINLAND
French (Canada)	fr-ca	CANADIAN FRENCH	CANADA
French (France)	fr	FRENCH	FRANCE
German (Germany)	de	GERMAN	GERMANY
Greek	el	GREEK	GREECE
Gujarati	gu	GUJARATI	
Hebrew	he	HEBREW	ISRAEL
Hindi	hi	HINDI	INDIA
Hungarian	hu	HUNGARIAN	HUNGARY
Icelandic	is	ICELANDIC	ICELAND
Indonesian	id	INDONESIAN	INDONESIA
Italian (Italy)	it	ITALIAN	ITALY
Japanese	ja	JAPANESE	JAPAN
Kannada	kn	KANNADA	INDIA
Korean	ko	KOREAN	KOREA
Latvian	lv	LATVIAN	LATVIA
Lithuanian	lt	LITHUANIAN	LITHUANIANA
Malay (Malaysia)	ms	MALAY	MALAYSIA
Malayalam	ml	MALAYALAM	
Marathi	mr	MARATHI	
Norwegian	no	NORWEGIAN	NORWAY
Oriya	or	ORIYA	

Table 16–4 Oracle HTML DB Globalization Codes

Language Name	Language Code	NLS_LANGUAGE	NLS_TERRITORY
Polish	pl	POLISH	POLAND
Portuguese (Brazil)	pt-br	BRAZILIAN PORTUGUESE	BRAZIL
Portuguese (Portugal)	pt	PORTUGUESE	PORTUGAL
Punjabi	pa	PUNJABI	
Romanian	ro	ROMANIAN	ROMANIA
Russian	ru	RUSSIAN	
Slovak	sk	SLOVAK	SLOVAKIA
Slovenian	sl	SLOVENIAN	SLOVENIA
Spanish	es	SPANISH	SPAIN
Spanish (Mexico)	es-mx	MEXICAN SPANISH	MEXICO
Swedish	sv	SWEDISH	SWEDEN
Tamil	ta	TAMIL	
Telugu	te	TELUGU	
Thai	th	THAI	THAILAND
Turkish	tr	TURKISH	TURKEY
Ukrainian	uk	UKRAINIAN	UKRAINE
Vietnamese	vi	VIETNAMESE	VIETNAM

A

Available Conditions

A condition is a small unit of logic that helps you control the display of regions, items, buttons, and tabs as well execute processes, computations and validations. When you apply a condition to a control or component, the condition is evaluated. Whether the condition passes or fails determines whether the control or component displays.

You can specify conditions by selecting a condition type when you create the control or component (that is, the region, item, button, or tab) or by making a selection under the conditional display attribute.

See Also: "Understanding Conditional Rendering and Processing" on page 6-9

Conditions Available in Oracle HTML DB

The following table describes some commonly used conditions. To view a complete listing of all available conditions for a given component, click the flashlight icon to the right of the Conditional Display Type list. Shortcuts to common selections appear directly beneath the Type list. If your condition requires an expression, type it in the appropriate field.

Table A–1 describes the conditions available in Oracle HTML DB.

Table A–1 Available Conditions

Condition	Description
Current Language != Expression 1	Verifies the language setting in which the client browser is not currently running. Evaluates to true if the current language is contained within the string entered in Expression 1.
Current Language = Expression	Verifies the language setting in which the client browser is currently running. Evaluates to true if the current language matches the value entered in Expression 1.
Current Language is contained within Expression 1	Determines whether the browser current language is contained within a string. Evaluates to true if the current language matches the string entered in Expression 1. For example, to check if the current language is either en-US or en-UK, choose this condition and enter the following string in Expression 1: `en-us,en-uk`
Current Language is not contained within Expression 1	Verifies the application's current language is not contained within a specified string. Evaluates to true if the current language is not contained within the string entered in Expression 1.
Current page != Expression 1	Evaluates to true if the current page does not equal the page you specify in Expression 1.
Current Page != Page Submitted (this page was not the page posted)	Determines if the specified page was not posted. Evaluates to true if the current page does not match the value entered in Expression 1.
Current page = Expression 1	Evaluates as true if the current page is in a list of pages found in Expression 1.
Current page = Expression 1	Evaluates to true if the current page equals the page you specify in Expression 1.
Current Page = Page Submitted (this page was posted)	Verifies the whether the specified page was posted. Evaluates to true if the current page matches the value entered in Expression 1.
Current Page is contained within Expression 1 (comma delimited list of pages)	Verifies if the current page is part of the list of pages you specify in Expression 1. To check if the current page is in either page 1, 2, 3 or 4, select this condition type and enter the following string in Expression 1: `1,2,3,4`
Current page is in Printer Friendly mode	Only displays certain page components when the user has selected printer friendly mode. If the current page is in printer friendly mode, then the condition evaluates to true. Use `f?p` syntax to specify printer friendly mode.
Current page is not in Printer Friendly mode	Hides page components when printer friendly mode is selected. Use `f?p` syntax to specify printer friendly mode. **See Also:** "Using f?p Syntax to Link Pages" on page 6-20 for more information on `f?p` syntax
Current Page not in Expression 1 (comma delimited list of pages)	Verifies if the current page is not part of the comma separated list of pages specified in Expression 1.

Table A–1 Available Conditions

Condition	Description
Exists (SQL query returns at least one row)	This condition is expressed as a SQL query. If the query returns at least one row then the condition evaluates as true. For example: `select 1 from emp where deptno = :P101_DEPTNO` This example references item `P101_DEPTNO` as a bind variable. You can use bind variables a within an application processes and SQL query regions to reference item session state. If one or more employees are in the department identified by the value of `P101_DEPTNO` then the condition evaluates as true. **See Also:** "About Bind Variables" on page 6-18 for more information on bind variables
Never	This condition type is hard wired to always fail. It is useful in temporarily preventing components (such as regions, buttons, or items) from being rendered on a page, or to prevent processes, computations and validations from running.
NOT Exists (SQL query returns no rows)	This condition is expressed as a SQL query. If the query does not return any rows, it evaluates as true.
PLSQL Expression	A PL/SQL expression is any expression in valid PL/SQL syntax that evaluates to true or false. For example: `nvl(:MY_FLOW_ITEM,'NO') = 'YES'` If the value of `MY_FLOW_ITEM` is YES then the condition evaluates as true. Otherwise it evaluates as false.

PLSQL Function Body returning a boolean	The body of a PL/SQL function that returns true or false. For example:
	`BEGIN`
	`IF :P1_DAY = 'MONDAY' THEN`
	`  RETURN TRUE;`
	`ELSE`
	`  RETURN FALSE;`
	`END;`

Table A–1 Available Conditions

Condition	Description
Request != Expression 1	REQUEST is an internal attribute that tracks of how a page is submitted. By default, when a page is submitted, the value of the application attribute REQUEST is set according the name of the object that caused the page to be submitted. For a regular button, REQUEST is set as the name of the button (such as CANCEL or SAVE) not the label of the button. For buttons of type Item or those that appear inline with other items, there is an attribute called REQUEST. You can also set request using f?p syntax. For example, the event could be when a user clicks a button or selects a tab menu. Depending upon the event, you can perform different operations by referencing the value of the REQUEST application attribute. This condition evaluates as true if REQUEST does not equal the value entered in Expression 1. **See Also:** "Understanding URL Syntax" on page 6-19, "REQUEST" on page 6-28, and "Understanding the Relationship Between Button Names and REQUEST" on page 7-42
Request = Expression 1	This condition is the opposite of `Request != Expression 1`. This condition evaluates as true if REQUEST equals the value entered in Expression 1. From PL/SQL you can also reference the application attribute using the following syntax: `v('REQUEST')` **See Also:** "Understanding URL Syntax" on page 6-19, "REQUEST" on page 6-28, and "Understanding the Relationship Between Button Names and REQUEST" on page 7-42
Request is contained within Expression 1	REQUEST is an internal application attribute that tracks of how a page is submitted. By default, when a page is submitted, the value of REQUEST is set according to the event that caused the page to be submitted. For example, the event could be when a user clicks a button or selects a tab. Depending upon the event, you can perform different operations by referencing the value of the REQUEST application attribute. Use this condition to specify a list of allowed requests (such as SAVE or UPDATE) in Expression 1. The condition evaluates to true if the value of REQUEST is contained in the list. **See Also:** "REQUEST" on page 6-28, and "Understanding the Relationship Between Button Names and REQUEST" on page 7-42
Request is not contained within Expression 1	This condition is the opposite of `Request is contained within Expression 1`. Evaluates to true if the value of the REQUEST is not contained within Expression 1. **See Also:** "REQUEST" on page 6-28, and "Understanding the Relationship Between Button Names and REQUEST" on page 7-42

Table A–1 Available Conditions

Condition	Description
SQL Expression	SQL Expressions are evaluated as a WHERE clause in a SQL statement. For example suppose your expression is `:MY_ITEM = 'ABC'`. The HTML DB engine processes the following: `select 1 from dual where :MY_ITEM = 'ABC'` This condition evaluates to true if a row is returned.
SQL Reports (OK to show the back button)	Use this condition for reports having pagination. It automatically determines when it is appropriate to include a button that pages back in the result set.
SQL Reports (OK to show the forward button)	Use this condition for reports having pagination. It automatically determines when it is appropriate to include a button that pages forward in the result set.
Text in Expression 1 != Expression 2 (includes &ITEM substitutions)	Use this expression to compare two expressions containing strings. Either expression may contain references to session state using `&MY_ITEM` syntax. **See Also:** "Using Substitution Strings" on page 6-22 for more information on `&MY_ITEM` syntax
Text in Expression 1 = Expression 2 (includes &ITEM substitutions)	This condition is the opposite of `Text in Expression 1 != Expression 2` (includes `&amp;ITEM` substitutions). Compares two expressions containing strings. Either expression may contain references to session state using the `&ITEM.` syntax. To check if the item `F100_P2_DAY_DATE` equals "Wednesday", select this condition enter the following in Expression 1 and Expression 2: ▪ Expression 1: `F100_P2_DAY_DATE` ▪ Expression 2: `Wednesday` **See Also:** "Using Substitution Strings" on page 6-22 for more information on `&MY_ITEM` syntax
User is authenticated (not public)	Verifies whether the current user was authenticated using one of the built-in authentication schemes or a custom authentication scheme. See Also: "Providing Security Through Authorization" on page 10-17 for more information on authentication
User is the public user (user has not authenticated)	The public user is defined as an application attribute. To set the public user for a specific application, navigate to the Application Builder home page and click the edit link corresponding to your application. A public user is a user used for multiple users. Sometimes applications have pages that are public and thus require authentication and log in. This condition returns true if the user is the public user (that is, the user is authenticated as themselves or some other user not equal to the public user identified in the application attribute Public User. **See Also:** "About Session Management" on page 7-33
Value of Item in Expression 1 != zero	Verifies if the value of the item in Expression 1 does not equal zero.

Table A–1 Available Conditions

Condition	Description
Value of item in Expression 1 = Expression 2	Compares the value of an item with a specific string. Comparisons using this condition are case sensitive. For example, to verify whether the value of an item `F100_P2_WORD` is contained within the string "the quick brown fox", enter the following in the Expression 1 and Expression 2 fields: ■ Expression 1: `F100_P2_WORD` ■ Expression 2: `the quick brown fox`
Value of Item in Expression 1 = zero	Verifies if the value of the item in Expression 1 does equal zero.
Value of item in Expression 1 contains no spaces	Evaluate to true if the value of the item specified in Expression 1 contains no spaces.
Value of Item in Expression 1 is alphanumeric	Evaluates to true when the string in Expression 1 contains only alphanumeric characters.
Value of Item in Expression 1 is contained within colon delimited list in Expression 2	Use this condition type to check whether a certain string is contained within the value of a session state item. Verifies whether the string specified in Expression 1 is contained in the value of the item specified in Expression 2.
Value of Item in Expression 1 is NOT contained within colon delimited list in Expression 2	Evaluates to true when the value specified in Expression 1 contains a string that lists elements delimited by colons. To check if the item `F100_P1_TODAY` is either "Monday", "Tuesday", or "Wednesday", select this condition and enter the following in Expression 1 and Expression 2: ■ Expression 1: `P1_TODAY` ■ Expression 2: `Monday: Tuesday:Wednesday`
Value of Item in Expression 1 is NOT NULL	In Expression 1, enter the name (upper case) of the application or page item. Evaluates as true, if the current cache value of the item is not null and has a value. If not, the condition evaluates as false.
Value of Item in Expression 1 is NULL	Evaluates as true if the item in Expression 1 has no value.
Value of Item in Expression 1 is NULL or zero	Evaluates as true if the item in Expression is either NULL or zero.
Value of item in Expression 1 is numeric	Evaluates to true if the value of the Item in Expression 1 is numeric.
Value of user preference in Expression 1 != Expression 2	This condition is the opposite of `Value of user preference in Expression 1 = Expression 2`. Evaluates to true if the name of the user preference specified in Expression 1 is not equal to the string in Expression 2.
Value of user preference in Expression 1 = Expression 2	Verifies the value of a user preferences. Evaluates to true if the name of the user preference specified in Expression 1 is equal to the string in Expression 2.
When any item in comma delimited list of items has changed	Evaluates to true when the value of any non NULL session state item in the list of items specified in Expression 1 has changed.
When any item in comma delimited list of pages has changed	Evaluates to true when the value of any non NULL session state item in the list of pages specified in Expression 1 has changed.

Table A–1 Available Conditions

Condition	Description
When any item in current application has changed	This condition passes when the value of any non NULL session state item in the current application has changed.
When any item in current page has changed	Evaluate to true when the value of any non NULL session state item in the current page has changed.
When any item in current session has changed	Evaluates to true when the value of any non NULL session state item in the current session has changed.
When cgi_env DAD_NAME != Expression 1	This condition is the opposite of `When cgi_env DAD_NAME = Expression 1`. Checks for the DAD (Database Access Descriptor) that is being used in the URL to call the current page in the application and compares it to Expression 1. Evaluate to true, when the DAD is not the same as Expression 1.
When cgi_env DAD_NAME = Expression 1	Checks for the DAD (Database Access Descriptor) that is being used in the URL to call the current page in the application and compares it to Expression 1. Evaluate to true, when the DAD is the same as Expression 1.
When cgi_env HTTP_HOST != Expression 1	This condition is the opposite of `When cgi_env HTTP_HOST = Expression 1`. Checks for the value of the CGI environment variable `HTTP_HOST` that is the value returned by `owa_util.get_cgi_env ('HTTP_HOST')`. Evaluate to true, when this value is not equal to the string in Expression 1.
When cgi_env HTTP_HOST = Expression 1	Checks for the value of the CGI environment variable `HTTP_HOST` that is the value returned by `owa_util.get_cgi_env ('HTTP_HOST')`. Evaluate to true, when this value is equal to the string in Expression 1.
When cgi_env SERVER_NAME != Expression 1	This condition is the opposite of `When cgi_env SERVER_NAME = Expression 1`. This condition checks for the value of the CGI environment variable `SERVER_NAME`, that is the value returned by `owa_util.get_cgi_env ('SERVER_NAME')`. Evaluate to true, when this value is not equal to the string in Expression 1.
When cgi_env SERVER_NAME = Expression 1	This condition checks for the value of the CGI environment variable `SERVER_NAME`, that is the value returned by `owa_util.get_cgi_env ('SERVER_NAME')`. Evaluate to true, when this value is equal to the string in Expression 1.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X