Oracle Application Express Best Practices

An Oracle White Paper
January 2006

ORACLE

Oracle Application Express Best Practices

Introduction 3
Application development Objectives 3
Development 4
  Database Design 4
    Enforcing Integrity Constraints 4
    Populating Primary Keys 4
  Development Productivity 5
    UI Defaults 5
    Reference Commonly Used Strings Indirectly 5
    Page Component Reuse 5
    Use Wizards to your Advantage 6
    Error and Exception Handling 6
Deployment 7
  Organizing Workspaces 7
  Deploying Applications 7
    Managing Database Objects 8
    Managing Static Files 8
    Making Applications Portable 9
User Interface consistency and usability 10
  Separate Style and Layout from Data and Application Logic 10
  Maintaining User Interface Consistency Across Multiple Applications 11
Security 12
  Security Begins in the Database 12
  Managing Users 12
  URL Tampering 13
  Cross Site Scripting 14
  Protecting against SQL Injection Attacks 15
  Protecting Report Columns 16
Performance 17
  Use Bind Variables 17
  Choose Pagination Style Wisely 19
  Use Declarative Logic when Possible 19
  Use of Redirects 19
Conclusion 20

Oracle Application Express Best Practices

INTRODUCTION

Oracle Application Express is a tool for quick development of web applications on an Oracle database. This paper provides a collection of best practices for the development and deployment of applications with Oracle Application Express.

APPLICATION DEVELOPMENT OBJECTIVES

There are a many characteristics that contribute to a web application being successful. An application that is:

- Easy to develop
- Easy to deploy
- Easy to maintain
- Easy to use
- Fast
- Secure

has a good chance of being successful This paper offers guidelines for developing applications that exhibit these characteristics. What's described in this paper should not be considered hard and fast rules. Rather, they are recommendations and best practices based on experience with developing and running Oracle Application Express applications in production.

DEVELOPMENT

Oracle Application Express is a highly productive tool, allowing you to build and deploy a fully operational reporting and data entry applications on the Oracle database in a matter of hours, sometimes minutes. Because it's so flexible, there often are several ways to achieve the same goal. The recommendations in this section help you achieve your goal in the most efficient way.

Database Design

Enforcing Integrity Constraints

It's difficult if not impossible to build an application based on a relational database without a proper database design. A proper database design includes referential integrity and uniqueness constraints guaranteeing the consistency and logical soundness of your data. Even if you believe you have sufficient validations or referential integrity rules enforced in the application, you often cannot guarantee that the database isn't accessed through other means. Although a complete survey of database design is beyond the scope of this paper, the advice is to exploit the innate capabilities of the database to maintain the accuracy and integrity of your data. Use primary keys, foreign keys, not NULL constraints and check constraints where appropriate.

Populating Primary Keys

Oracle Application Express includes wizards to create data entry forms on a database table. The wizard creates all the necessary UI controls and processes to provide insert, update and delete functionality. If you're using this form wizard and the automated DML process that it generates, it is often helpful to use a database trigger and a sequence to populate the primary key column. For example, if you have a table T with a primary key column ID, and a sequence MYSEQ, you could create a trigger MY_TRIGGER as follows:

```
create or replace trigger MY_TRIGGER before
insert on T for each row
begin
  if ( :new.ID is NULL ) then
     select MYSEQ.nextval
        into :new.ID
       from dual;
  end if;
end;
```

Development Productivity

UI Defaults

UI (user interface) Defaults in Application Express enable you to assign default user interface properties to a table, column, or view within a specified schema. These UI properties are used for screen elements such as field labels, format masks, column headers and column alignment. When you create a form or report using a wizard, the wizard uses this information to create default values for region and item properties.

Getting the UI defaults set to your taste beforehand can save you time during development because you eliminate the need to make many small adjustments to UI properties.

Reference Commonly Used Strings Indirectly

If your application frequently references on or more strings that are likely to change during the development of the application or afterwards, use substitution strings. Substitution strings are static variables defined at the application level. The value of these variables can be referenced throughout the application using the syntax &STRING_NAME. For example if your application references a product name frequently, but that name is subject to change, you could define a substitution string PRODUCT_NAME and reference it in a region as follows:

```
Welcome to the &PRODUCT_NAME. store!
```

With proper use of substitution strings, every time your product name changes, you need only make a change in one place.

Page Component Reuse

If a page component such as a region with static HTML or a list must appear on multiple pages in your application, consider using page 0. Page 0 is a page in your application that will never be run directly by the end user but whose components are considered for rendering by the engine on every page view. Components placed on page 0 can selectively excluded from other pages using conditions.

Use Wizards to your Advantage

Application Express provides several wizards that generate fully functional application components saving a lot of time over building them manually. What's unique about Application Express is that by using a wizard you do not sacrifice control or flexibility. Even though components and controls are generated for you, they can always be edited to modify the way they look or behave. The form on a table wizard is particularly useful, because in addition to providing a fully functional form capable of handling inserts, updates and deletes, it also provides lost update detection through optimistic locking.

Other wizards that can help speeding up the development process are the quick application wizard and the wizard to create a wizard. The quick application wizard generates a complete data entry reporting and analysis application on a single database table. The "wizard wizard" is named this way because it generates a wizard, a multi page data collection user interface. The wizard generates a series of pages with a complete with form fields, controls and page flow to navigate from page to page using Next and Previous buttons.

You are strongly encouraged to experiment with the wizards available in Application Express before you set out to build application functionality from scratch.

Error and Exception Handling

A Application Express application commonly executes PL/SQL logic in anonymous blocks or packages and stored procedures, for example to insert or update data in the database. If the PL/SQL called raises an exception, it should be dealt with gracefully. The recommended way to alert the user of this error without breaking the flow of the application is to assign the error message to an Application Express item in the event an exception is raised. This item can be displayed on the page calling the process that raised the exception. Finally, a conditional branch can be used to loop the user back to the page that produced the error allowing him to make changes and submit the page again.

Example:

Page 5 contains a data entry form whose values will be inserted into the database using a procedure called MY_PROC. If the insert is successful, the user should be taken to Page 1. If an error occurs, the user should be given the opportunity to correct this error. To display a user friendly error message on page 5, create an item called P5_ERROR_MSG. Define the after submit process as follows:

```
BEGIN
  MY_PROC (:P5_ITEM1, :P5_ITEM2);
  :P5_ERROR_MSG := null;
EXCEPTION when others then
  :P5_ERROR_MSG := 'Error in insert, please
correct values and try again';
END;
```

The error message can be displayed on page 5 in an HTML region using the following source:

```
&P5_ERROR_MSG.
```

When no exception is raised, P5_ERROR_MSG will be NULL and thus no message will appear on the page.

Finally, to make sure the user has an opportunity to correct the error that caused the exception, create a conditional branch, branching to page 5 when the value of P5_ERROR_MSG is not NULL and branching to page 1 when it is NULL.

DEPLOYMENT

Organizing Workspaces

In Application Express, a workspace allows one or more developers to develop applications, on database objects in one or more schemas. As a rule workspaces should be organized such that they contain applications that are related to each other. That is, applications developed by a common group of developers operating on a common set of data and schemas.

Deploying Applications

In Application Express, an application is comprised of the following components

- Database objects
- Application Express application definition
- Images (optional)
- Cascading style sheets (optional)

To deploy an application to a different database than the one it was developed on, you must deploy the application definition and the database objects it depends on. Optionally, you may need to deploy images and cascading style sheets (CSS).

Depending on requirements for the availability of the application, the size of the user population and other factors you can decide how sharp the lines need to be that divide the development environment from the production environment. For certain applications, it's acceptable to combine the development server with the deployment server, as long as the end users understand that sometimes the application is not available or briefly broken.

Other applications may require two (development and production) or even three (development, test and production) servers. In Application Express applications can be moved from environment to environment using an export and import facility for application definitions.

If you only have one piece of hardware available to run the database and Application Express, you can still separate the development version of an application from its production version by using two workspaces accessing separate schemas. One workspace will be used by developers and the other will be the workspace in which the application is deployed in production.

Managing Database Objects

If you deploy your application to a workspace and schema separate from the one in which it's developed, database objects may have to be moved with the application. If you are using Oracle Application Express to execute all database definition language (DDL) you have two options for applying changes in the deployment environment. One option is to manage your DDL in scripts in the SQL Workshop. The SQL Workshop allows you to edit, create, export and import scripts. If you capture all DDL in scripts in the SQL workshop, you can install database objects in a deployment environment by importing and running these scripts.

A second option is to use the export DDL functionality available in the SQL Workshop as well. Using this feature, you can export the DDL in the form of a script for any database object in your schema, regardless of how it was created. Re-creating these objects in production is then simply a matter of importing the generated script.

Managing Static Files

Application Express applications may reference additional external files such as cascading style sheets (CSS), images and Javascript libraries. These images can

either be placed on the file system of the server that runs the web server or, when they are uploaded using Application Express's application builder, can be stored inside the database.

A benefit of managing files such as images and CSS in the database with the application builder is that they can be exported and imported into another Application Express environment or workspace using only a browser. However, storing related files in the database is recommended only when you have relatively few images and when the application is not expected to endure high request throughput. When images, CSS and Javascript libraries are stored in the database, a database connection is established each time the browser requests one of these files. In a high throughput application this may result in undue and unwanted strain on the database.

Making Applications Portable

To ensure application can easily be moved from one Application Express environment to another you should avoid hard coded references to values that may change between environments. For example, the application ID is often referenced in URLs throughout an application. The Application Express engine uses the application ID to uniquely identify the application fetched and run from the application repository. Rather than using the actual application ID as is done in the URL below

```
f?p=100:1:&APP_SESSION.:
```

it is recommended you use an indirect reference instead:

```
f?p=&APP_ID.:1:&APP_SESSION.:
```

Here, `&APP_ID.` is a built in substitution string that replaces the hard coded value `100`. Using this technique, no modifications have to be made to the application prior to deployment, even if the application ID changes.

Similarly, to ensure smooth deployment, references to images in templates, HTML regions or SQL queries should use substitution strings in stead of hard coded paths. Consider, for example, an application whose images are stored on the file system in the virtual path `/dev/images/` in a development environment. When the application is deployed to a different server, images are stored in the virtual path `/prod/images/`. Use substitution strings defined at the application level to avoid having to make changes throughout the application before deployment. For example, define an application level substitution string `IMAGE_PATH`, and give it a value of `/dev/images/` Then, reference it in a query as follows:

```
SELECT '<img src="&IMAGE_PATH.edit_icon.gif" />' icon,
        ename,
        job
  FROM  emp
```

To reference images in a template, also use this approach. For example, the Header section of a page template may look like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
<head>
#HEAD#
<link rel="stylesheet" href="&IMAGE_PATH.custom.css"
type="text/css">
<title>#TITLE#</title>
</head>
<body bgcolor="#FFFFFF" #ONLOAD#>#FORM_OPEN#
```

In this example, `custom.css` is a cascading style sheet stored on the file system. Using this technique, only one change has to be made to the value of the application level substitution string IMAGE_PATH during deployment.

USER INTERFACE CONSISTENCY AND USABILITY

Separate Style and Layout from Data and Application Logic

Application Express provides a vast selection of components to speed up the construction of your application, including:

- Reports
- Forms
- Lists
- Navigation bars

- Menus

Each of these components is rendered using a template allowing you to define its structure separate from its style.

To maintain a clean separation between the style, logic and data access, avoid programming your own implementations of these components unless it's strictly unavoidable. As much as possible, use built in components to render HTML rather than writing custom code that renders HTML. This way, it's easier to change the application's look and feel without affecting its logic. For example, in Application Express a report is the formatted result of a SQL query. To have the ename column from the EMP table appear in bold on a page, you could create a report using the following query:

```
SELECT '<strong>' || ename || '</strong>', job
  FROM emp
```

It's better to use a query that doesn't incorporate any markup and then use a template or an HTML expression applied to the result set to achieve the same goal.

When you use Application Express's declarative components, you get several benefits. First, you can develop the meat of an application without having to worry about style and layout. The formatting of a report can be added later by applying an appropriate report template. The report template can be added by you at a later time, or can be designed in parallel by another developer.

Maintaining User Interface Consistency Across Multiple Applications

It's common for organizations to mandate a consistent look among all Intranet or Internet applications. When maintaining a single look and feel across multiple applications is a requirement, use template subscriptions. Template subscriptions are references to master templates defined in a central place. A good practice is to define a reference application that contains master copies of templates. Other applications can then subscribe their templates to these master copies. Each time an update is made to a master template, changes can be pushed down to templates that subscribe to it.

SECURITY

Securing a web application can be a daunting effort about which many volumes have been written. The following is by no means a complete recipe for protecting your application, but rather a few recommendations on commonly discussed security issues.

Security Begins in the Database

Relying on your application alone to provide security and access control is not sufficient. Just because you have good locks on the front door of your house, you don't leave stacks of cash on the coffee table ready to be snatched by someone who climbed through the window. Instead, you put your cash in a safe.

Similarly, the application you are building may not be the only means to get to the underlying data in the database. Alternate methods such as ad hoc query tools or direct SQL*Plus access may exist. If you build proper access control in at the database level you can rest assured that you don't have to re-implement security policies for each application that accesses the data in it.

To implement security in the database, make sure that the schema used to parse SQL and PL/SQL only has access to data strictly required for the application. If access to some, but not all, data in schemas A, B and C is required, consider creating an additional schema D with views defined on top of the necessary objects in A, B and C.

To restrict access to rows within a table based on the user that is querying it, use Fine Grained Access Control (FGAC). Fine Grained Access Control, sometimes called Virtual Private Database (VPD), is a mechanism to restrict access to rows within tables or views based on policies. These policies modify the predicate or WHERE clause of a query based on the application context. The application context allows you to determine who initiated the query, at what time and from what application. Based on these facts, a WHERE clause can be constructed such that only appropriate and allowed data is returned to the user.

Using FGAC, security policies need only be defined once on a database object. Every application accessing the underlying database object simply inherits the access control restrictions from the database. There is no need to re-implement the same security in the application.

Managing Users

When building and deploying Application Express applications you have a choice as to where and how end user credentials are managed. By default, end users are defined as users in the workspace that the application is part of. When deploying

to a large user population, it is highly recommended you use external identity management solutions such as an LDAP server or a Single Sign-On infrastructure such as that provided by Oracle Application Server. Keeping user definitions and credentials centrally defined in the organization reduces the burden of password management for the application administrator as well as the end user. The application administrator does not have to help people reset their passwords as often and end users need to remember fewer usernames and passwords, since the same password is used to log in to multiple applications. Also, revoking access from all applications when a user leaves your organization is simplified when her credentials are centrally managed.

URL Tampering

Web based applications, including those developed in Oracle Application Express often pass values from one page to another through a URL. A clever enough user may observe this and override a value by typing his own value in the location field of his browser. For example, on a page that displays a form for editing an employee record the value in session state of a certain item may determine which record is displayed In Application Express, the session state of this item can be set in the URL containing the following:

```
f?p=&APP_ID.:1:::::P1_EMPID:1234
```

Where `100` is the application ID, `1` the page ID, `P1_EMPID` the session state variable and `1234` represents a primary key value in a table of employees.

When a user clicks on such a link it will appear in the browser's location field. At this point the user can experiment and modify the URL into:

```
f?p=&APP_ID.:1:::::P1_EMPID:5678
```

in the hope of retrieving the employee record with the primary key value 5678. The user in question may or may not be allowed to look at the details for employee 5678. If she is not allowed, appropriate authorization rules should be applied to the logic on the page and, even better, at the database level to prevent her from viewing this data. Obscuring the value passed in the URL through a form of encoding or encryption is not sufficient.

For additional protection, Oracle HTML DB 2.0 introduced a new feature called Session State Protection. Session State Protection is a built-in functionality that prevents hackers from tampering with the URLs within your application.

Cross Site Scripting

Cross site scripting, sometimes called XSS, is a security breach that takes advantage of dynamically generated Web pages. In an XSS attack, a web application is sent with a script that activates when it is read by an unsuspecting user's browser. The script may steal data or even session credentials and send it off to a hacker. These attacks are rarely done by the application developer, but it is the developer's responsibility to protect against them. Because dynamic web sites rely on user input, a user with bad intentions can input malicious script into the page by hiding it within legitimate requests. If the developer does not protect the application from sending a malicious script masquerading as innocent user input, a innocent user could be subjected to an attack by the script.

To protect against these attacks, first filter all user input for unexpected characters and malformed input. Secondly, filter all output to the browser to prevent content from being unintentionally rendered as HTML or script. Application Express provides numerous wizards that create application components which, in themselves do not expose the application to these security issues. However, as you extend the application with additional functionality and custom code, you must take continual precautions, for example, by doing an end-to-end analysis of each data element, from the time it is introduced as input until it is emitted for rendering by the browser. An example of the latter would be where an application item named P1_ITEM is emitted on a page in a PL/SQL dynamic content region using the statement:

```
htp.p(:P1_ITEM);
```

If the value of `P1_ITEM`, perhaps in the usual case, established by some internal application logic, were instead introduced to the page by a hacker using the following URL:

```
f?p=APP:PAGE:::::P1_ITEM:<some malicious
javascript>
```

the emitting page would send the malicious script to the browser to do its damage. The developer should be aware of this possibility and filter the output used in the page. The recommended practice is to use functions such as `htf.escape_sc`, provided with the mod_plsql web toolkit.. This function converts tags and characters such as `<` and `&` to entity references `<` and & For example:

```
htp.p(htf.escape_sc(:P1_ITEM));
```

Protecting against SQL Injection Attacks

A SQL injection attack is a form of attack on a database-driven web site in which the attacker executes unauthorized SQL commands by taking advantage of flaws in its code. To prevent this type of attack don't include arbitrary user input in a SQL query run by your application. Consider the following anonymous block used as the source of a report. Here a query is dynamically built based on user input, stored in an Application Express item called `P1_USER_INPUT`. According to the PL/SQL logic below, If the user provided a value for the item, it will be appended to the query.

```
DECLARE
  q varchar2(4000);
BEGIN
q := 'SELECT ename, job, sal
  FROM emp
 WHERE deptno = 20 ';
IF :P1_USER_INPUT is not NULL then
   q := q || :P1_USER_INPUT;
END IF;
  return q;
END;
```

Assuming the value of P1_USER_INPUT is "`AND SAL < 2000`", the query executed will be:

```
SELECT ename, job, sal
  FROM emp
 WHERE deptno = 20 AND SAL < 2000
```

The results of this query will contain the names, job descriptions and salaries of all employees in department 20 whose salary is lower than 2,000. If, however the user types "OR 1=1" in the P1_USER_INPUT item, the query will read:

```
SELECT ename, job, sal
  FROM emp
 WHERE deptno = 20 OR 1=1
```

Now, the results will be the entire contents of the emp table! Avoid appending arbitrary user input to a query executed by the application unless your objective is to allow full ad hoc access to the underlying table. Instead, strictly control the structure of the query and carefully incorporate user input.

Protecting Report Columns

Reports in Application Express can be defined to show different columns to different users, based on a user's privileges. Authorization schemes are centrally defined access control rules that can be applied to elements throughout the application. The access control rule can be defined using SQL or PL/SQL or built in declarative logic. For example, assuming you created a table with authorized users called PRIVILEGED_USERS, you could implement an authorization scheme based on an EXISTS query as follows:

```
SELECT 1
  FROM privileged_users
 WHERE username = :APP_USER
```

In this example, the username of the currently logged in user, automatically stored in the session state variable APP_USER, is compared to all entries in the PRIVILEGED_USERS table. The way the authorization scheme is defined, only if a match exists, will it allow access to whatever element it is applied to.

Authorization schemes applied to report columns allow you to define a report once for all users while at the same time enforcing access control rules.

PERFORMANCE

Oracle Application Express's architecture is such that the majority of the processing occurs inside the database. The Application Express engine itself imposes little overhead, depending on the speed of the processors as little as 0.04 seconds of CPU usage per page request. Of course, as queries for reports and instructions for the Application Express engine to perform processing are added to a page, demand for CPU time increases. The following are recommendations to minimize load on the database as you develop your application.

Use Bind Variables

Before the Oracle database executes SQL queries or PL/SQL code, it goes through a series of steps that include parsing, name resolution, security checks and optimization. To avoid repeating these steps unnecessarily when the same SQL or PL/SQL is executed frequently, the database uses a shared pool or a library cache that stores pre-"compiled" SQL and PL/SQL. During execution of a query, the database will first look in this cache to see if it has already been run and if a version of it can be reused. To make queries as reusable as possible, you should use bind variables. Bind variables are place holders in SQL or PL/SQL to which values are assigned during execution. Consider for example this SQL:

```
SELECT empno, sal
  FROM emp
 WHERE empno = 1234
```

After this query has been executed, it will be available in the shared pool for reuse, subject to ageing routines that clean up this section of memory when space is needed. Of course, this particular query can only be reused if an application happens to execute the exact same query. That is, a query whose goal it is to get the employee number and salary of the employee with employee number (EMPNO) 1234.

Now, consider the same SQL with a bind variable:

```
SELECT empno, sal
  FROM emp
 WHERE empno = :empno
```

This query can be reused for any employee number because the actual value will be assigned when the query is executed.

Ensuring queries can be reused from the shared pool is critical for the scalability of a database application. Executing a pre "compiled" query, ready to go, from the shared pool takes less time than running it as if it were a brand new, never before seen query. As a result your users will consume fewer valuable resources in the database required to ensure concurrency.

How do you ensure query reuse in your Application Express applications? You do it by using bind variables when including values from session state in your SQL and PL/SQL. As a rule, avoid using substitution strings in SQL and PL/SQL. Consider this SQL query:

```
SELECT '<a href="f?p=100:1:&APP_SESSION.">edit</a>'
        link,
        empno, sal
  FROM emp
```

While at first glance it looks correct and harmless, it is not a query optimized for reuse. In fact, it will only be reused by one individual user. Before this query is executed, Application Express substitutes the current user's session identifier for &APP_SESSION. The database will see the query as:

```
SELECT '<a href="f?p=&APP_ID.:1:9878768768759">edit</a>'
        link,
        empno, sal
  FROM emp
```

Since `9878768768759` is a unique session identifier, the above query will be a unique query unless it happens to be executes more than once by the owner of that session. The preferred way to construct this query is:

```
SELECT '<a href="f?p=&APP_ID.:1:' || :APP_SESSION ||
'">edit</a>'
        link,
        empno, sal
```

```
FROM emp
```

Here, the session identifier is referenced as a bind variable which will be assigned after the database has retrieved a previously executed identical query from the shared pool. Note that it's OK to use the substitution variable &APP_ID, since that will always have the same value, unless of course there are multiple copies of the same application (with different application IDs) running in the same database.

Choose Pagination Style Wisely

Reports created in Application Express can be automatically configured to provide pagination through a result set. A result set of, say, 5,000 rows can be divided in to pages of 25 rows each. Different styles of pagination exist, some using just Next and Previous links and numbers indicating the currently displayed row ranges, others using a select list allowing the user to jump to a specific range of rows.

When it's not a requirement for the user to know exactly how many rows there are in a result set, use a pagination style that doesn't compute the result set size up front. For example using a pagination style such as "Row Ranges X to Y" can conserve valuable database resources.

Use Declarative Logic when Possible

There are many places where Oracle Application Express provides an opportunity for a developer to construct custom logic. For example in conditions applied to rendering or processing, or in validations on form input. To increase developer productivity, but also to optimize performance, many pre-defined condition types and validations exist.

Since these pre-defined units of logic are essentially "burned in" to the Application Express engine, you are strongly encouraged to use them whenever possible to improve application performance. For example, to validate whether user input is numeric, you can write your own PL/SQL logic to implement a validation. Choosing the pre-built "Item is numeric" validation type takes less time and will ultimately result in better performance.

Use of Redirects

Most pages in an Application Express application go through two distinct stages as they are run. The first stage is the rendering stage during which the page is assembled based on its definition stored in the Application Express application repository together with other components and templates it references. Then, when the user triggers an event, such as the clicking of a button to insert data in the

database, the Application Express engine takes the page through the processing stage. In this stage, validations are run and processes are executed to modify session state or data in the database. Finally, branches are executed to take the user to another page (or the same page).

Certain user triggered events do not require processing. For example, clicking a Cancel button to navigate from a data entry form to another page often does not require any modifications to session state. For this scenario, Application Express offers a type of button that does not incur any processing and in effect provides a short circuit by navigating the user away from the current page using a URL redirect

CONCLUSION

Oracle Application Express is a highly productive declarative tool that doesn't sacrifice the flexibility for developers to have precise control over the behavior, style and layout of the application. This means that Application Express developers have choices in the development process.

In this paper I've discussed guidelines and best practices to ensure easy development and deployment of web applications with Oracle Application Express that are secure, perform well and predictably and use a consistent and easy to maintain user interface.

Oracle Application Express Best Practices
January 2006
Author: Sergio Leunissen

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com