ORACLE White Paper

Designer/2000
for Oracle Applications Users

IMPLEMENTING AND CUSTOMIZING RELEASE 10SC

VERSION 1.0: FEBRUARY, 1996

February 1996

Author: Mark Pirie

Contributors:

Table of Contents

Introduction

A successful application implementation involves significant investment beyond the initial purchase price, and there will be an ongoing cost through the life of the application. What is key is that this is done quickly, efficiently and in a way that protects the investment from future changes. It would be optimistic to believe that any package provides a 100% fit to requirements, even with generic features built in to the products. Furthermore it is certain that the requirements will change over time, be it changes in the organization or external changes such as new legislation. The aim of this document is to show how Oracle Designer/2000 can be used to store the application definition in a common repository where it can be used to help identify missing functionality, then aid the modeling, design and generation of customer specific extensions.

To maximize the cost/benefit of an application package through its lifespan there are some important questions to be asked when purchasing the application:

- **Does it support the business requirements?**
- **How easy is it to implement?**
- **How easy is it to build custom extensions?**
- **How flexible is it to changing business requirements?**
- **Will the implementation be protected from future application or technology changes?**

Oracle Applications are developed with a business model which meets core application requirements and has many generic structures and features which allow the applications to be adapted to specific customer requirements. In addition to this there is the added complexity of providing functionality specific to a geographic area, such as sales tax in the US and VAT in Europe. One of the biggest hurdles to overcome when implementing sophisticated applications like this is to understand exactly what is provided by the package. This could be understanding the data model, the features, the supported business processes, the security model, etc.

In reality many of these issues are equally relevant to the Oracle Applications development group who need to record, document and use a common repository for their development which is split across multiple teams. They currently use Oracle Designer/2000 to aid this process and to generate some of the core application screens.

In summary, the main benefit in using Designer/2000 for an applications implementation project is that the core package and any customization work is fully documented in the common repository. The information is then available throughout the life of the application to anyone using it or working on its maintenance. This protects the development investment when business processes or technology change, as there is a well defined understanding of the current implementation and the tools available to model and implement those changes at minimal cost.

Implementing Oracle Applications

Understanding the Business Processes

Before an organization can start implementing any package they must first understand their current business processes. In most cases this is something they must consider before they are in a position to select the correct package. The people in the organization that understand these processes are not normally from the IT department, but are the managers and employees who perform the tasks. The issue here is how we can record this information in a way that allows anyone to understand and modify the processes.

The **Oracle Process Modeller** is a very easy to use, graphical tool which allows people with little prior experience of modeling to record their business processes. Examples of business processes might be: order fulfillment, customer acquisition, customer service, hiring an employee. These processes often span several departments or organizations and completing the full model requires experience from groups of people who need to work together to paint the whole picture.

So how does the Process Modeller help those implementing and using Oracle Applications?

- It helps people to understand how their existing business processes work by providing a creative medium for recording and communicating the processes.
- The Models are stored in a common repository which allows the whole team to access them, and utilize them in any subsequent development.
- It allows the existing business models to be mapped on to the models supported by Oracle Applications, thus helping identify the correct implementation strategy.
- It helps identify extensions to the applications required to fulfill a particular customer requirement.
- It encourages teams to suggest ways of improving the processes.
- It communicates the improvements widely, and helps people to understand how their own work may be affected. The models can be used as a training aid to help roll-out the new applications. The multimedia capabilities of the modeller allow sound, video, images and text to be embedded, and this significantly improves the understanding of the processes.
- It supports the roll-out of the new processes.

It is a fact of life that over time the organization’s business processes are likely to change. By adopting this approach of documenting the business, it allows the processes to be modified and communicated as appropriate.

In the example below there is a simplified ‘Hire Person’ process model. The diagram has organization units listed down the left hand side with these continuing across the diagram in what are termed ‘swim lanes’. Modeled on the diagram are a series of process steps linked together using process flows. Using the model it is very easy to communicate the process with all members of the project team and identify missing steps or unproductive processes.

Once a process model is agreed on there is the task of mapping the process to the Oracle Applications. In the example the complete process is supported except there is no functionality in the core application for notifying security about producing a badge. The model shows that we need to resolve three things:

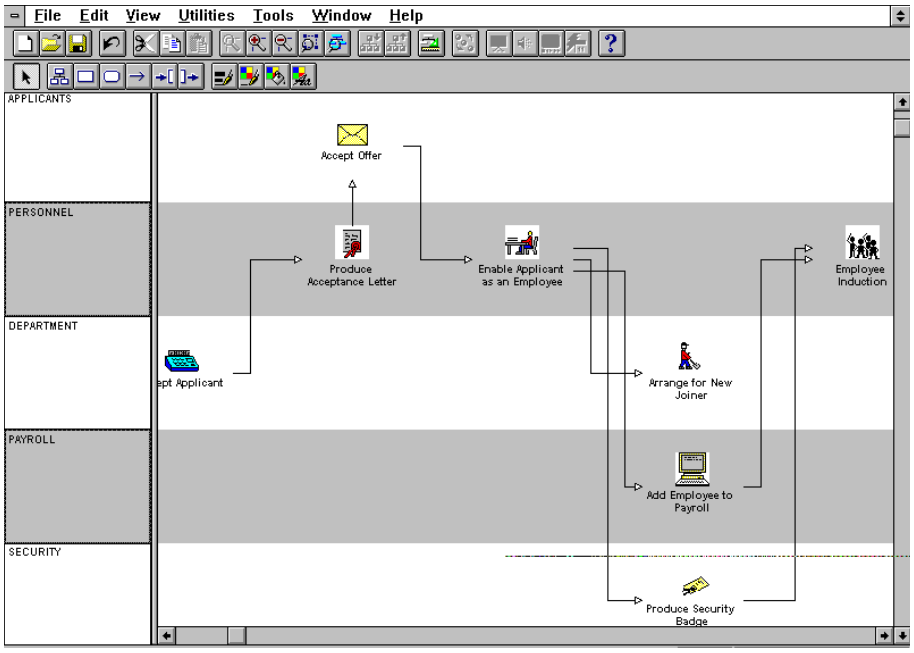
- How Personnel notify Security about the required badge?
- How the badge is produced?

- How the badge gets back to Personnel?

At this point there is more than one way to solve the problem, some possible examples are:

- This part of the process could be purely manual involving internal mail to send details between the organization units and a manually produced badge.
- The core applications standard features such as ‘Alerts’ could be used to automatically send mail from Personnel to Security when an employee is hired.
- An extension to the application could be generated to allow the security organization access to the employee details and produce the badge from the system.
- The process could be remodeled to allow personnel to produce the badges directly.

This very simple example shows that this technique can help adapt the customer process to best fit the Oracle Applications and identify any customization required to complete the implementation.



Application Server Design Recovery, Extension and Generation

One of the most difficult problems for anyone implementing or customizing complex applications is understanding the data model of the supplied package. The problem is compounded where there are existing bespoke or packaged applications which need to integrate with the new applications. Without a tool like Designer/2000 the only solution is to work from the on-line dictionary or from paper documentation, both of which are difficult and non-productive.

At the System Design level of the Designer/2000 product it is possible to fully record the complete application server and client design. We will discuss client side generation later in the paper, but this section will concentrate on how the tool allows server side application design to be recovered, extended and then the generated.

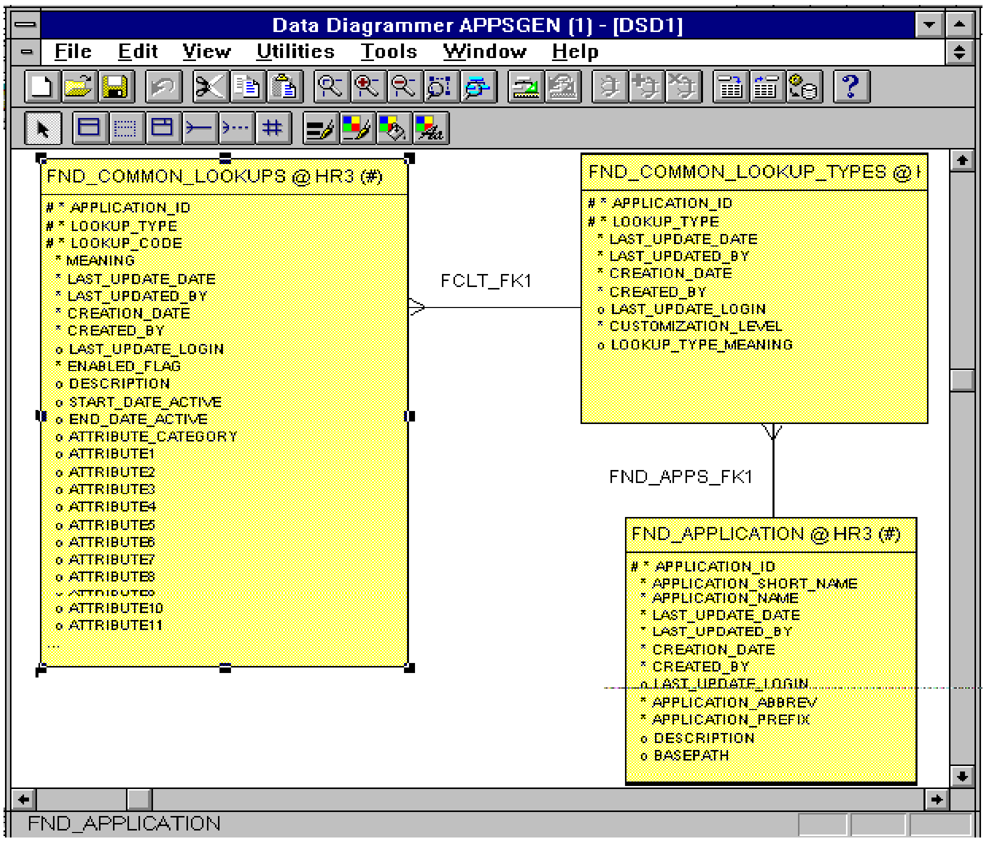
There are two options for building the data model for the core Oracle Applications:

1. Because Oracle Applications are built using Designer/2000 it is possible to get extract files containing the applications definitions for Oracle Financials, Oracle Manufacturing or Oracle Human Resources. These can be obtained through your Oracle contact or from the Oracle Applications development group.

2. The application design can be recovered directly from the installed Oracle Applications database using the Reverse Engineer Database utility. This allows all Oracle7 objects, including any procedural logic to be recorded in the Designer/2000 repository. After running the utility the repository will directly mirror what is implemented in the database. It is also possible to use this technique to recover the design of any bespoke or other packaged application which is based on an Oracle7 database.

Whichever route you take to complete the design recovery, it is important that information in the repository is checked for completeness. One important omission is that some of the Oracle Applications do not enforces foreign key or primary key constraints in the database, thus these cannot be recovered, and must be manually added using Designer/2000 to complete the model. This step does require a basic knowledge of the Oracle Applications data model and is simplified by the consistent table and column naming. It is strongly recommended that the Release 10 Oracle Applications Technical Reference Manual is obtained to complete this task as it details all the missing constraints.

Once the design has been recovered there are a number of diagrammers for graphically displaying the information. The Data Diagrammer is the key method for displaying and modifying data model. Below is an example diagram showing three table definitions which have been reverse engineered into the repository. Later in the paper there is an example how this part of the model is used as the basis for generating an screen.



Once the applications are fully recorded in the design level of Designer/2000 it is possible to start to extend and customize them. Again there are two routes:

1. Some development organizations are familiar with the concept of producing an information model for the business requirements they are developing as a database system. Some organizations represent that information model as an entity relationship model. We will not cover the principles and techniques of ER modeling here, but for those organizations it is possible to use the Table to Entity Retrofit utility to derive a ‘first-cut’ entity model from the physical definition tables. From

this starting point it is possible to model the application extensions and use the Database Design Wizard utility to migrate the model extensions back down to the design level.

2. The data model can be extended directly by adding and modifying tables, columns, views, etc.

Either of these routes will result in a complete and documented data model from which to run the server generator to generated the DDL to update the physical database. Below is a example of the generated DDL to implement the foreign key constraint on the table FND_COMMON_LOOKUPS.

```
REM
REM  This ORACLE7 command file was generated by Oracle Server Generator
REM  Version 5.5.8.0.0 on 06-FEB-96
REM
REM For application APPSGEN version 1 database HR3
REM
REM CONSTRAINT

PROMPT Adding FOREIGN Constraint To FND_COMMON_LOOKUPS Table

ALTER TABLE FND_COMMON_LOOKUPS ADD (
    CONSTRAINT FCLT_FK1
    FOREIGN KEY (LOOKUP_TYPE,
             APPLICATION_ID)
    REFERENCES  FND_COMMON_LOOKUP_TYPES (
             LOOKUP_TYPE,
             APPLICATION_ID)
)
```

The same approach can be used for reverse engineering other bespoke or packaged applications into Designer/2000. This provides the same benefits as for the core applications and makes it much easier to integrate the applications. Once the repository contains the whole business data model, decisions can be made about whether the individual applications should be fully integrated to share a common data model, or to keep them separate and build interfaces between them.

Generating Oracle Application Forms

The most commonly asked question when Oracle Applications users look at Designer/2000 is "Can it generate forms with the Oracle Application look and feel?". This section briefly describes the Oracle Applications development architecture and then looks in detail at how Designer/2000 can be used to implement many of the special features of Oracle Applications.

It is important to ask yourself what are you trying to achieve. I believe the answer should be to produce the required functionality in a productive, flexible and maintainable way, i.e. the resulting screens should look and feel like any other applications screen and allow the end user to perform their tasks efficiently. It is important is that we should not get too wrapped up in how the functionality is provided as long as meets the above requirements.

From a development perspective, producing a packaged application is more complicated than producing a bespoke solution as the package has to be generic, have the ability to run in multiple languages, be portable, etc. However, when customizing such a package many of these issues are not so relevant, and hence the implementation can be simplified.

The details given in this section assume that the reader has a working knowledge of Designer/2000 and in particular the Oracle Forms Generator.

Note that it is **very important** that the versions of Oracle Forms used in Designer/2000 and Oracle Applications are the same.

Oracle Applications' Architecture

Oracle Applications Release 10SC is the most recent version of Oracle's suite of packaged applications which provide a complete graphical user interface. The SmartClient architecture is based on application partitioning to separate all the user interface presentation and processing on the client and all the data oriented applications processing on the server.

In addition to using Oracle7 RDBMS, PL/SQL and Oracle Forms 4.5 the Oracle Applications are built using an enabling technology, Application Object Library (AOL). Application Object Library is an application in its own right which is used to provide standard features in all the other applications. It provides data-driven features which allow individual customers to customize the applications to their own requirements with no custom coding. Typical examples of such features are:

- Descriptive FlexFields and Key FlexFields
- User Menu Structures and Function Security
- Folder Forms

Note that because these features are part of the core applications they are protected from any changes when the applications are upgraded. The limitations of this approach is that the generic solution may not be ideal for a particular customer situation. If this is a problem the alternative approach is to model and generate the customization using Designer/2000, and generate again when the core application is upgraded.

Oracle Applications Standards

The Oracle Applications development team have developed the set of coding and user interface standards which are adhered to in the core applications.

Oracle Applications User Interface Standards (Part Number A31209)

Oracle Applications Coding Standards (Part Number C10070)

These standards rely on having Application Object Library (AOL) installed as they are supported by template forms, client-side libraries and server-side procedures provided by AOL. It is important that the generator templates that you build, reference and use these forms and libraries, as by doing this you will benefit from all the standard AOL features and isolate yourself from changes to the Oracle Applications standards.

Before either hand coding or generating any custom forms it is important that the developer has a good understanding of these documents and the standard building blocks they provide to simplify development. Before looking at how we would generate Oracle Applications' screens, let's look closely at the mechanism provided by AOL for simplifying hand-coding.

The Oracle Applications standards rely extensively on the object referencing capabilities of Oracle Forms. These capabilities allow objects to be reused across multiple forms, with changes to the master instance automatically inherited by forms that share the object. Additionally, these referenced objects provide flexibility for cross platform-support, allowing Oracle Applications to adhere to the look and feel of the platform they run on.

AOL provides a special form **APPSTAND.fmb** which contains the master copy of all referenced objects and is the key to development. It contains the following:

- Object group STANDARD_PC_AND_VA, which contains the Visual Attributes and Property Classes required to implement much of the user interface described in the standards.
- Object group STANDARD_TOOLBAR, which contains the windows, canvases, blocks and items of the Applications Toolbar. This group also contains other items which are required in all forms but are not necessarily part of the Toolbar.
- Object group STANDARD_CALENDAR, which contains the windows, canvases, blocks and items of the Applications Calendar.

Note there are additional objects in the APPSTAND form which are currently for internal use by Oracle Applications only, and their use by customers is not supported. Specifically the object group STANDARD_FOLDER which provides the structure for Applications folder forms.

APPSTAND is altered slightly on each platform to adjust properties such as button height, scrollbar widths and Toolbar locations. Application Object Library ships the correct APPSTAND file in the $AU_TOP/resource directory (or its equivalent) for the specified platform.

AOL also provides a starter form, TEMPLATE.fmb, for any form being developed by hand. (Note this is equivalent to the Oracle Forms Generator template form.) When hand-coding, the standards recommend copying this file, located in $FND_TOP/forms/US (or your language and platform equivalent), to a local directory and renaming it as appropriate.

TEMPLATE contains the following:

- Platform-independent references to object groups in the APPSTAND form (STANDARD_PC_AND_VA, STANDARD_TOOLBAR and STANDARD_CALENDAR).
- Platform-independent attachments of several standard libraries (FNDSQF, APPCORE and APPDAYPK)

- Several form-level triggers with required code. The text within these triggers must not be removed; however, frequently developers will need to add text before or after this text. A full list of these triggers is covered in the section on "Building a Generator Template".
- Package APP_CUSTOM, which contains default behaviour for window opening and closing events. This code usually requires modifying for each form.
- The Applications color palette, containing the two colors required by the referenced visual attributes ('canvas' and 'button'), 'pure' colors (such as 'black', 'white', 'blue' and 'red'). Note that a custom color palette is attached to specific canvas, if the canvas is dropped from the form the form reverts to the Oracle Forms default color palette. In the case of Applications development this will result in canvases turning 'yellow' instead of the intended 'canvas grey'.
- The TEMPLATE form contains sample objects that show typical items and layout cosmetics. These are provided as samples, and can be removed by deleting blocks: BLOCKNAME and DETAILBLOCK, window: BLOCKNAME and canvas: BLOCKNAME. When generating these are not required.

Application Object Library supplies a number of standard libraries which support the Applications standards. These are attached to the TEMPLATE form as described above and do the following:

- FNDSQF contains packages and procedures for message dictionary, flexfields, profiles, and concurrent manager. It also has various other utilities for navigation, multicurrency, WHO, etc.
- APPCORE contains the packages and procedures required of all forms to support the menu, Toolbar, and other required standard behaviours. Additionally it contains packages that should be called to achieve specific runtime behaviours in accordance with the standards, such as the way in which items are enabled, behaviours of specific types of windows, and the dynamic 'Special' menu. Finally it contains various other utilities for running totals, exception handling, messaging levels, etc.
- APPDAYPK contains the packages that control the Applications Calendar.

APPFLDR contains all of the packages that enable folder blocks. Note use of the folder technology outside of core development is not supported so do not attach this to any custom screen.

This section has described the standard development infra-structure across all the Oracle Applications. Each of the Applications is likely to have extended this philosophy to accommodate Application specific features or code. This could well mean that there will be template or client-side libraries which are specific to an individual application or group of applications. There may well be documentation detailing this, but if not, then look at some of the forms source and determine the application specific extensions and references. There may even be a template specific to that application.

Oracle Forms Versions

It is worth stressing again that the success of Oracle Application forms generation is dependent on the version of Oracle Forms being the same in Designer/2000 and Oracle Applications. **Under no circumstances must the version used in Designer/2000 be ahead of that in Oracle Applications.**

With considerable effort it is possible to succeed if Oracle Applications uses a more recent version of Oracle Forms than Designer/2000, if all the development is done with the Designer/2000 version and then Oracle Forms is used to upgrade to the Oracle Applications version. Note in this case it would be important to have a version of APPSTAND which was concurrent with Designer/2000.

Building a Generator Template

In most respects the generator template required for the generation of Oracle Applications screens is very similar to the TEMPLATE form used for the core development. If you want to develop a new template from scratch then I suggest that you copy TEMPLATE and modify this to achieve your generator template.

To simplify this task we have provided a starting point generator template which integrates with the applications APPSTAND reference form, the client-side libraries and contains the required standard triggers and program units. The form is APPSGEN.fmb and this will be shipped with the standard templates in Designer/2000 release 1.3, and can be obtained from the author until then.

So what does the APPSGEN template contain?

- References of the object groups STANDARD_PC_AND_VA, STANDARD_TOOLBAR and STANDARD_CALENDAR.
- Attached client-side libraries FNDSQF, APPCORE, CUSTOM, APPDAYPK.
- The four standard generator alerts.
- The canvas CUSTOM_PALETTE_CANVAS which has the standard Applications color palette attached. Make sure that this canvas is never removed from the template.
- The program units APP_CUSTOM, APP_MESSAGES, MSG_ALERT, SET_VISUAL_ATTRIBUTES.
- Through the referenced object group the template contains all the Applications standard visual attributes. These are not directly used by the generator and thus the template contains a set of CG$Visual Attributes which match the Applications visual attributes.
- Applications standard forms level triggers, see list below. Note that user logic can be added to these triggers, but they should not be removed.
    - KEY-CLRFRM
    - KEY-COMMIT
    - KEY-DUPREC
    - KEY-EDIT
    - KEY-EXIT
    - KEY-HELP
    - KEY-LISTVAL
    - KEY-MENU
    - ON-ERROR
    - POST-FORM
    - PRE-FORM
    - PRE-INSERT
    - PRE-UPDATE
    - WHEN-FORM-NAVIGATE
    - WHEN-NEW-BLOCK-INSTANCE
    - WHEN-NEW-FORM-INSTANCE
    - WHEN-NEW-ITEM-INSTANCE
    - WHEN-NEW-RECORD-INSTANCE
    - WHEN-WINDOW-CLOSED
    - WHEN-WINDOW-RESIZE

  User-Named Triggers

    - ACCEPT
    - CLOSE-THIS-WINDOW
    - CLOSE_WINDOW
    - LASTRECORD
    - QUERY_FIND
    - STANDARD_ATTACHMENTS
    - ZOOM

Generating Standard Applications Features

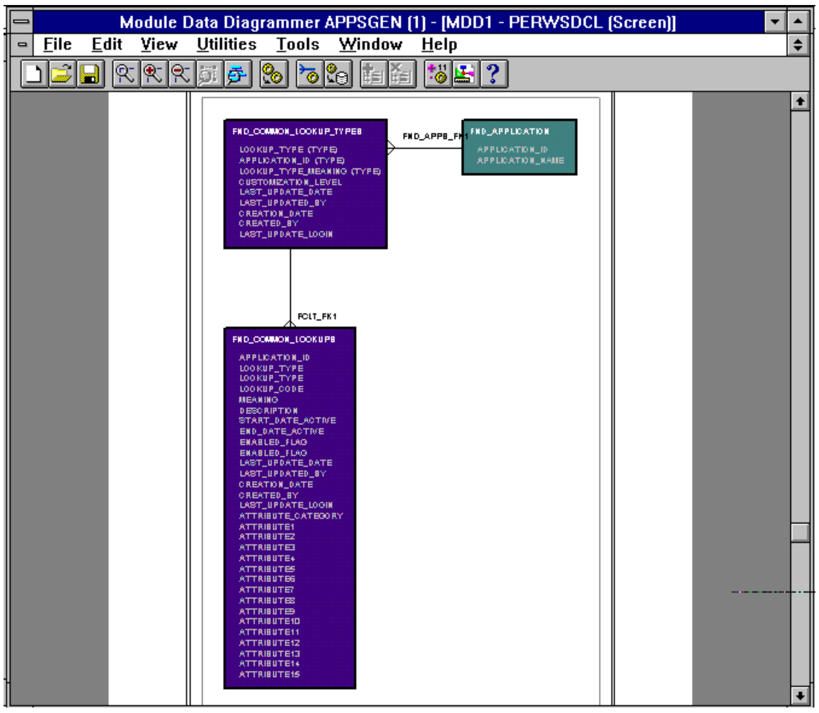
When using the Designer/2000 Oracle Forms Generator there are three key components to controlling the generated form:

- **Template Form** - The section ‘Building a Generator Template‘ covers how to build the basic applications template. The template is used to include standard objects and code into each generated form, and to act as a template for any generated objects. This will need to extending for some of the features described in this section.

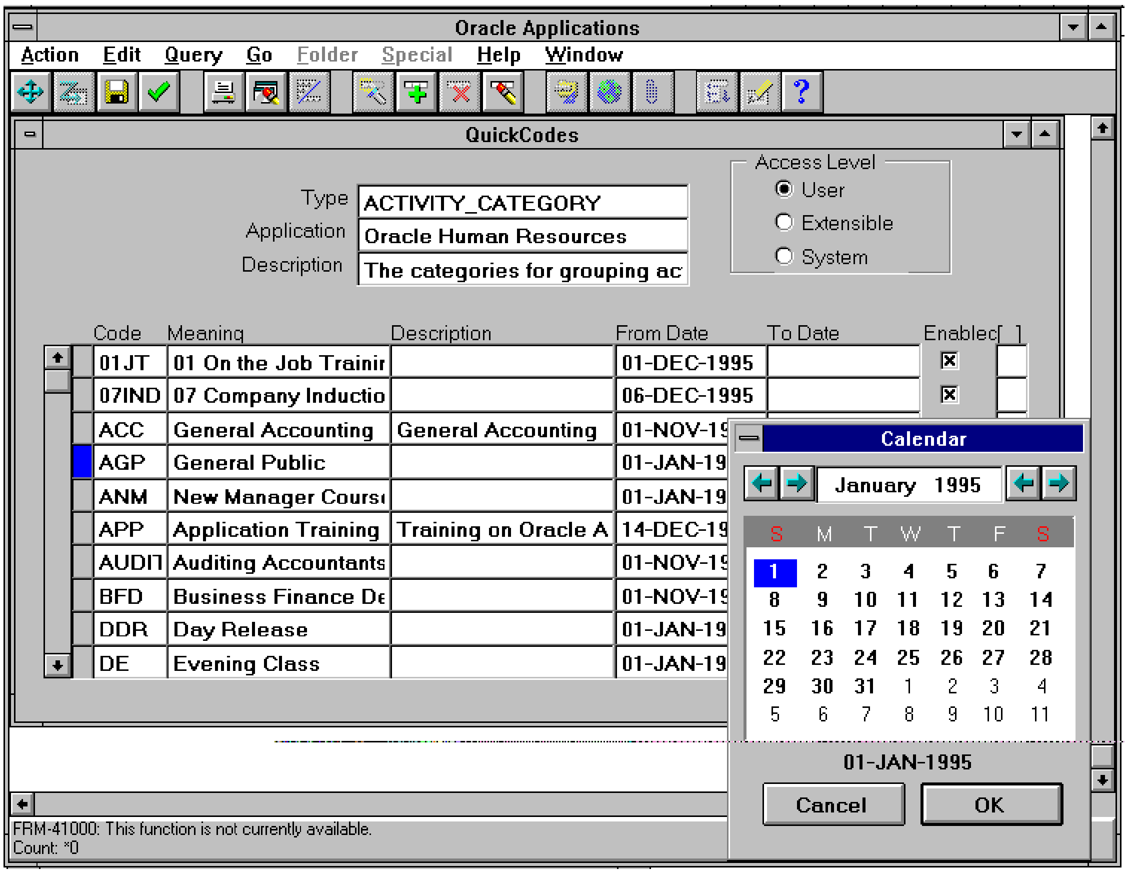
- **Preferences** - These control the look and feel of the generated form and act as a style guide for the application. The preferences can be set at different levels and it is important that the preferences are set at the highest possible level, i.e. Application Level. Doing this ensures that the standards implemented using the preferences are applied to all generated forms. It will be necessary to override some of the preferences at a level below Application level to tailor the layout for particular blocks, item groups, items, etc. The sensible approach to take here is to create a number of preference sets to implement standard features.

- **Module Definition** - This defines the structure of the module specifying the usage of the tables and columns and some basic layout information.

Modules are defined using the Module Data Diagrammer where the structure of the module is defined graphically. In the example below the module is a master-detail form with a single lookup, and shows which tables and columns are used. The look and feel of the resulting form is controlled by setting properties and preferences, and from the template form used during generation.



The form generated from the module defined above is the standard applications “QuickCodes” form. Having generated the form it is necessary to register it with Oracle Applications and then include it in the user’s menu. Below is the completed form running under Oracle Applications, which includes Application features such as Descriptive Flexfield, calendar, toolbar and current row indicator:



Type ACTIVITY_CATEGORY
Application Oracle Human Resources
Description The categories for grouping ac

Access Level: [✓ User] Extensible System

Code	Meaning	Description	From Date	To Date	Enabled
01JT	01 On the Job Trainir		01-DEC-1995		☒
07IND	07 Company Inductio		06-DEC-1995		☒
ACC	General Accounting	General Accounting	01-NOV-19		
AGP	General Public		01-JAN-19		
ANM	New Manager Cours		01-JAN-19		
APP	Application Training	Training on Oracle A	14-DEC-19		
AUDIT	Auditing Accountants		01-NOV-19		
BFD	Business Finance De		01-NOV-19		
DDR	Day Release		01-JAN-19		
DE	Evening Class		01-JAN-19		

January 1995

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

01-JAN-1995

FRM-41000: This function is not currently available.
Count: *0

Generation of Specific Application Features

In this section the intention is to give detailed information on how some of the standard Oracle Applications features can be implemented using the template, preferences and module definition. It is hoped that the examples given will show how any other feature may be tackled using Designer/2000.

Application Level Preferences

There are a number of preferences that should be changed from the default at Application level and never changed at a lower level.

Preference Name	**Preference Description**	**Value**
ITMBIF	Base item sizes on the font	N
ITMDPS	Item default prompt separator	<null>
ITMPIG	Prompt item gap	1
STFFMB	The name of the template form	appsgen.fmb
OTBAEN	Order template blocks at the end	Y
BLKTGS	Block level trigger execution style	AFTER
ITMTGS	Item level trigger execution style	AFTER
WINUSR	User primary window to display first generated page	N
HLPTYP	Type of Help System used	NONE

Descriptive Flexfields

Most Oracle Applications tables contain columns required to support Descriptive Flexfields and if required these can be included in new tables created as part of the customization. In this case it would be necessary to register the new Descriptive Flexfield through the Application Administrator. The normal columns are ATTRIBUTE1 to ATTRIBUTE(n) and a context column normally named ATTRIBUTE_CATEGORY. When displayed in a form the Descriptive Flexfield is a non-basetable item and the functionality is implemented using a number of standard AOL procedures. Designer/2000 requires the correct column usages defined and use of Descriptive Flexfield preferences to fully generate the items and logic to support this feature. The steps are as follows:

- Create non-displayed Detailed Column Usages (DCU) for ATTRIBUTE1 to ATTRIBUTE(n) and the context item ATTRIBUTE_CATEGORY.

- Create a displayed secondary DCU for the actual displayed Descriptive Flexfield and set the following properties:

Display Datatype:	Descriptive Flexfield	
Display Width	: 2	
Prompt	: [<3 spaces>]	(For multi-row block)
	: [|]	(For single row block where '|' is the post-prompt marker)

Note that for Descriptive Flexfields in a single row block the following preferences should be set at DCU level:

Preference Name	**Preference Description**	**Value**
ITMPPM	Item post prompt marker	|
ITMPIG	Prompt/Item gap	0

- Occasionally the AOL implementation requires, in addition to the standard Descriptive Flexfield, two associated item. These are the displayed description field and the hidden id field. If these are required, create a secondary DCU of Display Datatype 'Descriptive Flexfield', directly after the usage defining the displayed Descriptive Flexfield DCU. Set the usage flags (Insert, Update, Select, Nullify) all to 'No'. The 'Display' flag must be set, even for the hidden data id field, or the Display Datatype will be lost. The id item is distinguished from the description item by checking the 'Other' flag.

- Set the Descriptive Flexfield preferences as defined in the table below. All the preferences (except REGNDF) refer to the generation of a particular Descriptive Flexfield and the preferences are applicable at the DCU level. However, in many cases the a single Descriptive Flexfield will be generated in several forms and thus it is strongly advised that the preferences are defined at Table level. These preferences hold the values of the parameters which are passed to the AOL Descriptive Flexfield procedures. See the Oracle Applications standards for more detail.

Name	Preference Description	DescFlex Parameter
DFFNAM	Descriptive Flexfield - Field Name - Defines the actual name of the displayed Descriptive Flexfield item generated	N/A
DFNAME	Descriptive Flexfield - Name	DESC_FLEX_NAME
DFAPPL	Descriptive Flexfield - Application Short Name	APPL_SHORT_NAME
DFDESC	Descriptive Flexfield - Description Field Name	DESCRIPTION
DFSIDS	Descriptive Flexfield - Segments Ids Field Name	DATA_FIELD
DFDATE	Descriptive Flexfield - VDATE definition	VDATE
DFTITL	Descriptive Flexfield - TITLE definition	TITLE
DFPICK	Descriptive Flexfield - AUTOPICK definition	AUTOPICK
DFUSDB	Descriptive Flexfield - USEDBFLDS definition	USEDBFLDS
DFREAD	Descriptive Flexfield - READ_ONLY definition	READ_ONLY
DFLOCK	Descriptive Flexfield - LOCK_FLAG definition	LOCK_FLAG
REGNDF	Regenerate Descriptive Flexfield, controls whether or not the generator recreates triggers associated with Descriptive Flexfields when regenerating.	N/A

Implementing the WHO Columns

The WHO feature reports information about who created or updated rows in the Oracle Applications tables. It is important that this information is properly maintained by all modules which change the data as some of Applications upgrade technology relies on this information, plus this information is available to the end-user through the Application menu. If any table contains the standard WHO columns then any Detail Table Usage (DTU) based on that table should include non-displayed DCU's of these columns.

If new tables are added to the data schema you may wish to add these standard columns to have a consistent implementation across the whole model.

The updating of the WHO columns when a row is inserted or updated is done by the procedure "fnd_standard.set_who". The supplied template APPSGEN.fmb contains form level PRE-INSERT and PRE-UPDATE triggers which call this procedure. The standard columns are:

CREATED_BY	NUMBER(15)	NOT NULL
CREATION_DATE	DATE	NOT NULL
LAST_UPDATE_BY	NUMBER(15)	NOT NULL
LAST_UPDATE_DATE	DATE	NOT NULL

LAST_UPDATE_LOGIN NUMBER(15)

Setting Window Titles

The current version of the generator creates window titles which are of the format:
<Module Name>: Window(n). This is not standard in Oracle Applications and can easily be changed by taking the following steps:

- Add a line in the appsgen.fmb template PRE-FORM trigger to call a procedure ‘user_set_window_titles’

- Create a module specific library <Module Name>.pll, (you may have created one already for other features), and add a procedure ‘user_set_window_title’. In this procedure add the logic to set the window title for each of the windows, i.e.

  ```
  set_window_property(window_name, title, ‘<Window Title>‘);
  ```

- Set the module level preference MODLIB - Module specific library attachment to <Module Name>.pll.

Note that this technique allows you to use one generic template with module specific features, and this is a common theme to implementing Oracle Applications standard features. Once you use this technique you must make sure that every module generated with the APPSGEN.fmb template attaches a module specific library which contains the procedure, even if it contains a null statement.

In **Designer/2000 Release 1.3** the window title property can be set for the first Detailed Table Usage in each window defined. In most cases this will replace the mechanism above where the window title is a fixed character string. If the title needs to be more dynamic, then the above approach will still be valid.

Calendar

The Calendar is a standard object that allows the selection of date and time values from a calendar. It also allows the developer to specify validation rules ensuring that only valid dates can be selected. The List or Edit function invokes the calendar on any date item.

The template APPSGEN.fmb contains all the components required to implement the Calendar. Add the following logic to the APPSGEN.fmb template:

- When the user enters any date item the standards say that the ‘List Lamp’ should be enabled. To do this logic needs to be added to the PRE-FORM trigger. The APPSGEN.fmb template contains a LOV named ENABLE_LIST_LAMP which is used for this. The PRE-FORM trigger should contain the procedure call ‘user_set_date_lov’.

- Create a module specific library <Module Name>.pll, (or you may have created one already for other features), and add a procedure ‘user_set_date_lov’. In this procedure add the logic to set the item property ‘lov_name’ to ENABLE_LIST_LAMP for each date item in the form, i.e.

  ```
  set_item_property(item_name, lov_name, ‘ENABLE_LIST_LAMP’);
  ```

- The item name can easily be determined if you understand the standard way the generator names the generated items, else this can be determined post generation by viewing the form using the Oracle Forms Designer.

- The default form level KEY-LISTVAL in the template APPSGEN.fmb must be modified to call to the procedure ‘cg_call_calendar’ before any existing logic in the trigger. Note there is already a call to APP_STANDARD.EVENT('KEY-LISTVAL') in this trigger and thus the trigger logic should be:

  ```
  if user_call_calendar(:system.cursor_item) = False

      then APP_STANDARD.EVENT('KEY-LISTVAL');
  ```

```
            end if;
```

- Add the procedure ‘user_call_calendar’ to the module specific library and include the logic:

```
FUNCTION user_call_calendar (item_ name char)
RETURN boolean IS
BEGIN
  if    item_name = '<BLOCK.ITEM_NAME>'
  or    item_name = '<BLOCK.ITEM_NAME>'
    then calendar.show;
         return(true);
   else  return(false);
 end if;
END;
```

  **Note BLOCK.ITEM_NAME is the block and name of each date item in the form, and should be in uppercase.

- Set the module level preference MODLIB - Module specific library attachment to <Module Name>.pll.

In **Designer/2000 Release 1.3** the Calendar will be automatically generated. There will be a new preference USECAL which when set to ‘Y’ enables the calendar generation. For each date item which needs the calendar to be invoked, define on the Detailed Column Usage PL/SQL text the standard notation “COMPLEX:” a call to a standard program unit in the template which calls the calendar procedures in the Oracle Applications APPDAYPK library. The “COMPLEX:” call can pass parameters to the program unit if more sophisticated logic is required than the standard ‘calendar.show’ procedure call. If the calendar call is the basic calendar.show, this can be directly entered for the PL/SQL text, i.e. COMPLEX:calendar.show.

If the preference is set and the PL/SQL text is defined the generator will automatically assign the LOV ‘ENABLE_LIST_LAMP’ to the date item and generate an item level KEY-LISTVAL trigger to invoke the calendar. It will be necessary to remove any code added to implement the feature before release 1.2.

Current Row Indicator

All Oracle Applications multi-row blocks have a one character item at the start of the row which is used to indicate the current row. For the current row this item is colored blue. This can be easily implemented by taking the following steps:

- Add a display only secondary DCU for the current row indicator item, making this the first displayed item in the row. Set the following DCU properties:

Display Datatype	:	Char
Display Width	:	1
Prompt	:	<null>

- Add a line in the appsgen.fmb template PRE-FORM trigger to call a procedure “user_current_row_indicator”

- Create a module specific library <Module Name>.pll, or you may have created one already for other features, and add a procedure ‘user_current_row_indicator’. In this procedure add the logic to set the following properties for each current row item in the generated form, i.e.

```
set_item_property('<item_name>', visual_attribute, 'DISABLED_TEXT');

set_item_property('item_name', current_record_attribute, 'CURRENT_RECORD');
```

- Set the module level preference MODLIB **-** Module specific library attachment to <Module Name>.pll.

In **Designer/2000 Release 1.3** the current record indicator feature can be automatically generated following these steps. Note that this replaces any implementation described above.

- Add a display only secondary DCU for the current row indicator item, making this the first displayed item in the row. Set the following DCU properties:

```
Display Datatype:  Current Record

Display Width    :  1

Prompt           :  <null>
```

- Add the two named visual attributes to the template

  - CG$CURRENT_RECORD, set to the same properties as the 'CURRENT_RECORD' visual attribute already referenced from the APPSTAND template.

  - CG$OTHER_RECORD, set to the same properties as the 'DISABLED_TEXT' visual attribute already referenced from the APPSTAND template.

Inter-Block Navigation

The standard method for inter-block and inter-window navigation is using buttons. These can be generated using the Designer/2000 button generation feature. The on-line help details how this is done for the simple case using the “COMPLEX:” procedure call notation. With more sophisticated applications it may be necessary to do more complicated processing than simply specifying GO-BLOCK, such as checking for outstanding commits, context sensitive navigation, etc.; This can be handled in the same way as the simple case, but with the procedure referenced in the COMPLEX call being held in the module specific library. This allows processing to be tailored for each button in the generated form.

As part of the forms navigation model it may be necessary to alter the default generated NEXT-BLOCK and PREVIOUS-BLOCK processing. If hot keys are mapped for these functions, by default, the action will cause navigation from one block to another based purely on their sequence. This is often undesirable, particularly when a master block has multiple child blocks with no natural order. It would probably be more appropriate to disable next-block from the master to any of the child blocks and rely on buttons for the navigation. There are two approaches to solving this:

- The simplistic approach is to disable default navigation by defining KEY-NXTBLK and KEY-PRVBLK form level triggers with a “null;” statement.

- The more complete solution is to add a procedure ‘user_set_block_nav’ in the module specific library. In this procedure add the logic to set the next block and previous block properties for each block. Add a call in the PRE-FORM trigger of the APPSGEN.fmb to call “user_set_block_nav”. The procedure logic to do this is as follows:

```
set_block_property('<block_name>', next_navigation_block, '<next_block_name>';
```

or set_block_property('<block_name>', previous_navigation_block, '<previous_block_name>';

Note that the next and previous block name can be set to the current block name to prevent any navigation.

Oracle Applications Messaging

Oracle Applications has its own message dictionary where all the messages required by the applications are held external to the forms and programs. The dictionary will already hold all the message strings for the delivered applications and can be used to add any new messages required to support your customizations. Using the AOL Message Dictionary, you can:

- Define of standard messages that can be shared between modules
- Provide a consistent look and feel for messages within and across all your applications
- Define flexible messages that can include context-sensitive variable text using tokens
- Change or translate text of your messages without regenerating or recompiling your application code

The Oracle Applications Coding Standards describe the mechanism for defining these messages and the API procedures supplied to retrieve, set up and display them.

By default the Oracle Forms Generator hard codes messages into the PL/SQL of the generated forms. There are two types of messages which the Generator creates:

1. Messages which the Generator creates itself, such as: ‘Press [Delete Record] to confirm deletion’.
2. Messages which the developer specifies in the repository against objects such as constraints.

To enable you to access the message text, you can specify that the Generator is to ‘externalize’ generated messages in a separate library file (called ofg4mes.pll) that is attached to the form at generate time. This library is supplied with the Forms Generator and contains a message string for all messages of type (1) above. For Oracle Applications each of the message strings in the library should be replaced with a call to display an AOL message dictionary message. The syntax for this will depend on whether the message had tokens, but would be something like:

```
FND_MESSAGE.SET_NAME(‘Application’,’Message Name’);

FND_MESSAGE.SET_TOKEN(‘Token Name’,’Value’,’Translate Flag’);

FND_MESSAGE.SHOW
```

For Type (2) messages these are best handled in the message procedure defined in the template form. For each message defined in the repository only specify the application and the Message Dictionary message name concatenated, i.e. FNDMESSAGE_NAME.

To externalize the message text and generate take the following steps:

1. Define new messages in AOL Message Dictionary.
2. Create a suitable procedure in the template form (see later example).
3. Modify the supplied library ofg4mes.pll.
4. Set the preference MSGSFT to the name of the procedure in the template form.
5. Generate the form.

Overleaf is an example of the code required in the template procedure to handle all the message calls to AOL Message Dictionary.

```
PROCEDURE APP_MESSAGES (
errt in char,            /* message type */
rftf in boolean,         /* raise form_trigger_failure ? */
errm in char,            /*message*/
p1   in varchar2 default '',
p2   in varchar2 default '',
p3   in varchar2 default '',
p4   in varchar2 default ''
) is      /* message parameters */
msg_string     varchar2(200);
application    varchar2(3);
alert_is       alert;
alert_button   number;
BEGIN
  /* Get the message if its a generator one */
  cg$message.get_message(errm,msg_string,p1,p2,p3,p4);
  if (msg_string is null)  /* ie not a normal ofg45 message */
  then
   application := substr(errm,1,3);
  msg_string  := substr(errm,4);
  fnd_message.set_name(application,msg_string);
  if ((errt = 'F') or (errt = 'E'))
  then
    fnd_message.error;
  elsif (errt = 'W')
  then
    if fnd_message.warn then
       null;
    else
       null;
    end if;
  elsif (errt = 'I')
  then
    fnd_message.show;
  else
    /* shouldn't ever get to here - but just in case */
    message(msg_string);
  end if;
 else
  if (errt = 'F')
  then
    alert_is := find_alert('CFG_SYSTEM_ERROR');
  elsiF (errt = 'E')
  then
    alert_is := find_alert('CFG_ERROR');
  elsif (errt = 'W')
```

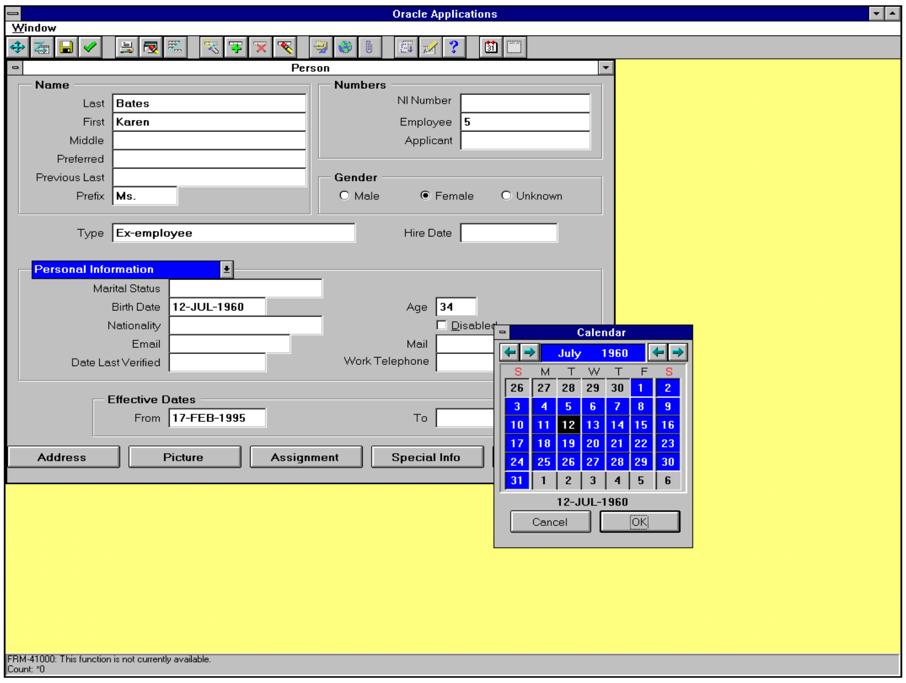
```
    then
      alert_is := find_alert('CFG_WARNING_A');
    elsif (errt = 'I')
    then
      alert_is := find_alert('CFG_INFORMATION');
    else
      message(msg_string );
    end if;
     if (errt IN ('F','E','W','I'))
     then
       set_alert_property(alert_is,ALERT_MESSAGE_TEXT,msg_string );
       alert_button := show_alert(alert_is);
     end if;
  end if;
  if (rftf)
  then
    raise form_trigger_failure;
  end if;
END;
```

Alternate Regions

In **Designer/2000 Release 1.3** we will introduce support for generating Oracle Applications alternate regions. In some of the busy or complex forms there is a requirement to split the display of a record over multiple canvases. The Application standards say that these canvases should work like a Window95™ tab, but use a poplist to switch between the canvases.

To implement this feature an item group should be created for each canvas required and the items placed into the appropriate item group. These item groups should have the Layout Style property set to ‘Stacked’ to differentiate them from the standard item groups. During generation all the standard item group layout preferences apply to the stacked groups, and the item group titles become the options in the generated poplist which controls the navigation between canvases.

In the example below the ‘Personal Information’ poplist is the control for the generated alternate region.



Name
Last: Bates
First: Karen
Middle:
Preferred:
Previous Last:
Prefix: Ms.

Numbers
NI Number:
Employee: 5
Applicant:

Gender
Male [✓ Female] Unknown

Type: Ex-employee
Hire Date:

Personal Information
Marital Status:
Birth Date: 12-JUL-1960
Age: 34
Nationality:
Disabled
Email:
Mail:
Date Last Verified:
Work Telephone:

Effective Dates
From: 17-FEB-1995
To:

Calendar
July 1960

S	M	T	W	T	F	S
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

12-JUL-1960

FRM-41000: This function is not currently available.
Count: *0

Extending the Application Beyond Forms

Oracle Forms is the prime tool for producing the Oracle Applications user interface, but it is not the only one available when customizing the applications. Designer/2000 provides generation capabilities for Oracle Reports, Oracle Graphics, MS Help and Web applications. A large part of the implementation and customization effort goes on providing specific reporting functions, be it standard formal reports or MIS style ad-hoc queries.

The Oracle Reports Generator and Oracle Graphics Generator work on the same principle as the Oracle Forms Generator. They are driven from a module definition in the Designer/2000 and use templates and preferences to control the style of the generated report or graphic. Having the full applications data model in the repository, including model extensions, simplifies greatly the learning curve when trying to understand the data model for creation of this type of reporting. The Graphical interface makes it much easier to understand the table and view structure and the keys linking them together, and the common repository make it possible for the model to be fully documented so that such information can be shared amongst the development team.

The great benefit of a multi-user repository driven development tool is it makes it much easier to understand the application structure, and hence develop the reporting modules.

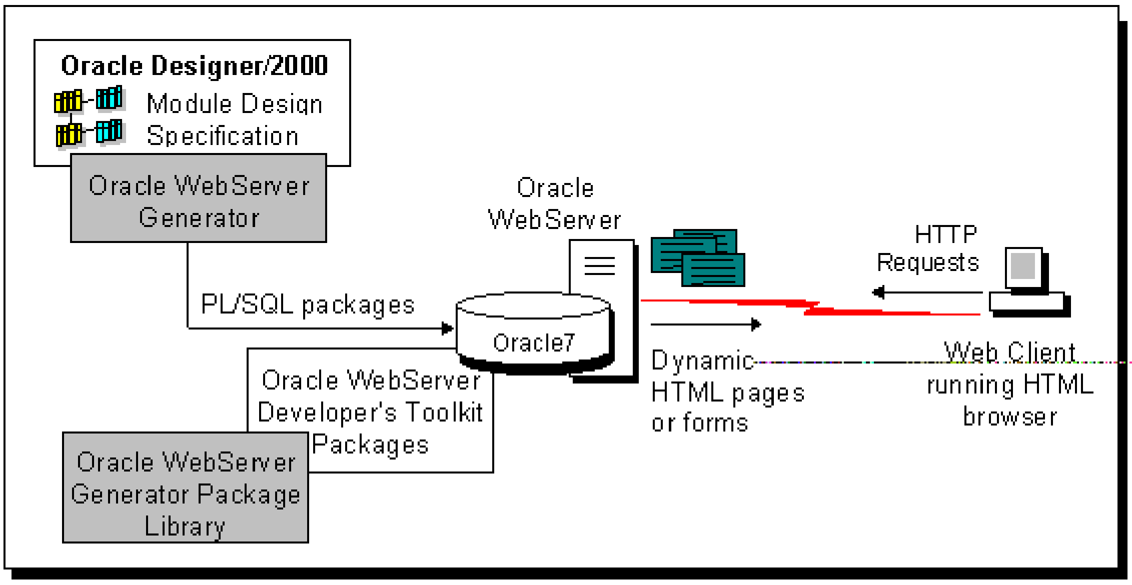
In this paper I don't intend to cover in any greater detail how to use the these generators as they are well covered by the on-line documentation. Oracle Applications does not have rigid standards for this type of module so the issues are the same as on any other development project.

Oracle WebServer Generator

The Worldwide Web is one of the most rapidly growing technologies. It enables companies and individuals to publish information by creating Hyper-Text Markup Language (HTML) documents and publishing the Universal Resource Locator (URL) of these documents. These documents can then be viewed by any of the industry standard Web Browser such as Oracle PowerBrowser or Netscape, with the document access controlled by a configuration file or network firewall.

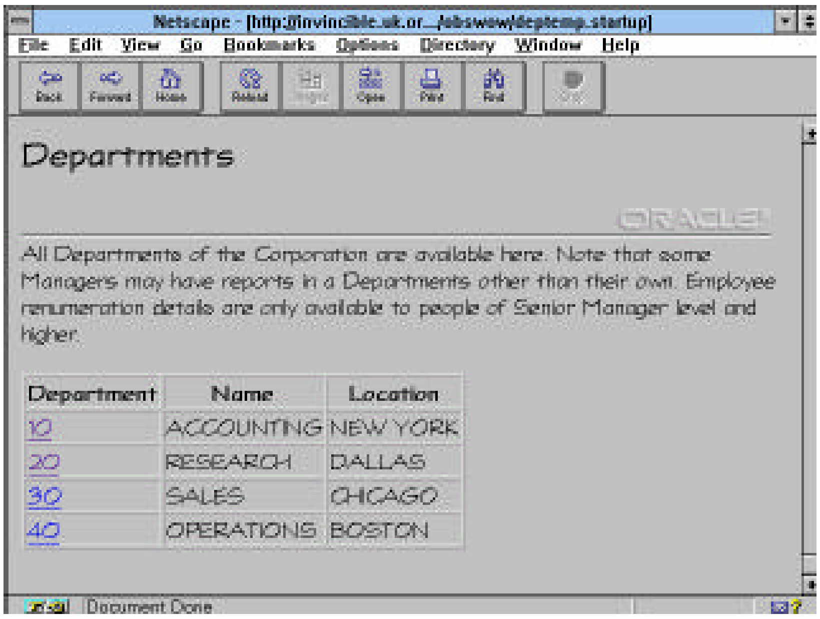
More recently the Web technology is being regarded as a viable tool for application development, offering an alternative user interface using widely available technology. The launch of Oracle WebServer has enabled the Web applications to easily access the Oracle7 Server. This opens the door for Oracle Applications users to use this technology to customize and extend the core applications.

New to Designer/2000 is the Oracle WebServer Generator (WAG). This utilizes the Designer/2000 repository and relies on the same modeling techniques used by the Oracle Forms generator. Thus it offers the ability to generate and deploy Web applications from definitions already built and deployed using Developer/2000. During generation WSG creates a set of PL/SQL packages, which are then installed into the Oracle WebServer database. The PL/SQL packages contain the logic to access the application data and formatting statements to display the Web pages when accessed using a browser. Preferences control the look and feel of the generated application and can be customized to suit particular requirements.


Oracle Designer/2000
Module Design Specification
Oracle WebServer Generator
PL/SQL packages
Oracle WebServer
Oracle7
Oracle WebServer Developer's Toolkit Packages
Oracle WebServer Generator Package Library
HTTP Requests
Dynamic HTML pages or forms
Web Client running HTML browser

One of the major benefits of deploying application extensions using the Web is that the application is held on the server and thus removes the need for mass software distribution whenever the application changes. The many peripheral users of the information retained in the Oracle Applications can now be accessed without the requirement for a full client side install of the standard applications. So if you want to give all employees access to basic HR information such as department, title and telephone number it can be done cheaply, using this solution. It is equally feasible to make some of the information available, externally, to your customers, suppliers, etc.;

Below is a simple example of generated Web page that displays information retrieved from an Oracle database.


Netscape - [http://invincible.uk.or.../obswow/deptemp.startup]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
ORACLE
Document Done

Departments

All Departments of the Corporation are available here. Note that some Managers may have reports in a Departments other than their own. Employee renumeration details are only available to people of Senior Manager level and higher.

Department	Name	Location
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON