Scott Systems (EPSO)
Reports Technical Standards

Author: **M.V.R.Murthy**

Creation Date: July 23, 1999

Last Updated: XXX 0, 0000

Control Number: 1

Version: 1

Approvals:

Rajesh Gupta	

ORACLE®
Enabling the Information Age™

Document Control

Change Record

Date	Author	Version	Change Reference
23-Jul-99	M.V.R.Murthy	1	No previous document

Reviewers

Name	Position

Distribution

Copy No.	Name	Location
1	Library Master	Project Library
1		Project Manager
1		
1		

Contents

Introduction

Purpose

This document describes the technical standards that Scott Systems Pvt. Ltd.,(EPSO) will follow when customizing Oracle Applications Reports and development of new reports for extensions of Oracle Applications using the tool Reports 2.5.

While the document Dev-std.doc (Technical Standards , Scott-EPSO) describes the cosmetic standards applied to Reports , this document describes the Technical standards to be used in development of reports using Reports 2.5 tool , and hence should be read in conjunction with the Dev-std.doc (Technical Standards , Scott-EPSO). This document makes no attempt to clarify the concepts of Reports 2.5 tool and as such assumes that the reader is familiar with Reports 2.5 and other Oracle development tools.

This document is targeted at the Scott-EPSO technical consultants involved in building new and customized Oracle Applications reports.

Background

The information in this document has been defined ,

1.As the result of discussions between Scott-EPSO technical consultants.

2.On the basis of study of various Oracle Applications Reports by Scott-EPSO technical consultants.

Scope & Application

The standards in this document cover the Build phase and will primarily affect tasks in the Module Design and Build process of AIM. The document applies to all Oracle Applications Reports development and customizations undertaken by Scott-EPSO.

Related Documents

1. *Technical Standards for Scott-EPSO (Dev_Stds.doc)*
2. *Oracle Applications Developer's Guide – Release 11*
3. *Oracle Applications Flexfield's Guide – Release 11 .*
4. *Reports 2.5 Reference Manual*
5. *Microsoft Visual SourceSafe V 5.0 Reference Manual*

Notational Conventions

You should be familiar with the notational conventions listed in the following table.

Convention	Explanation
`File--New--Report`	Indicates a selection of submenu item Report from the New submenu of the File menu.
`Data/Selection:`Repeat	Indicates a selection of Repeat from the Data/Selection tab of a property sheet.
Font Change	Indicates text to be entered exactly as shown.
UPPERCASE	Indicates command, column, parameter, field, boilerplate, and anchor names.
Initial Caps	Indicates table, menu, query, group, frame, and repeating frame names, as well as property sheet names.
Bold	Indicates items within a menu, buttons, or tabs on a property sheet.
Italics	Indicates options for Oracle Reports settings

Report Coding Standards

Naming Conventions

The naming conventions for various Reports 2.5 Objects should be as specified in the following table,

Object Type	Name Format	Explanation	Comments
Lexical Parameter	LP_D	D=Description	
Bind Parameter	P_D	D=Description	
Query	Q_D	D=Description	Description should reflect the query selection. Example : Q_Payments
Group	G_D	D=Description	Description should be the column Name on which the Group is based
Formula Column	CF_D	D=Description	Follow same guidelines as function naming.
Summary Column	CS_D_XXX	D=Source of the Column XXX=Function of Calculation	Example : CS_sal_sum
Placeholder Column	CP_D	D=Description	
Rectangle	RC_D	D=Description	Description should reflect the location of the object. Example : RC_Orders_Line_ header
Frame	M_D	D=Description	Description should describe the objects it is enclosing.
Boilerplate Text	B_D	D=Description	

Style and Structure

The cosmetic standards applicable would be as elaborated in *Dev_Stds.doc*.
To ensure a standard layout and ease in design always enable **`View--Grid`**, and **`View--Grid Snap`** in the layout editor.
Ensure a top and bottom margin of 0.5''. In the case of documents like Purchase Order or Invoice that could require filing , provide a left margin of 0.5".
Always turn **Confine mode** to ON unless an object has to be explicitly moved out of the enclosing object. Also , always turn the **Flex Mode** to OFF unless some objects have to

be explicitly pushed in the push path of an object while the object is being re-sized or moved.

Suggestion: When using Tool bar in Layout Editor position the cursor on the Tool icon and double click , this enables the tool for multiple operations as against a single click which enables the Tool for a single operation only.

The following specific settings should be followed in case of **Character** and **Bitmap** reports ,

Character Mode Reports :

i. Set **Tools--Tools Options : Runtime Parameters:**Mode as *Character*

ii. Report Module--**Properties :** Set the following properties in the **Character Mode** tab

Report Width x Height *80 x 66*

Use Character units in Designer : *Checked*

iii. In Layout Editor **View--View Options--Rulers**

Set the **Units** *Character cells*

Grid Spacing *1*

Number of snap points per Grid Spacing *1*

Let the **Character Cell Size (points)** be Horizontal 6.5 , Vertical 12.

iv. Set the **Format--Font** to Courier – Regular 9

Bitmap Reports :

i. Set **Tools--Tools Options : Runtime Parameters:**Mode as *Bitmap*

ii. Report Module--**Properties :** Set the following properties in the **Character Mode** tab

Use Character units in Designer : *unchecked*

Report Width x Height *8.5 x 11*

In case it is a customization of a character mode report to bitmap report convert the report width and height using the formula ,

Width = Old width in Character units * 8.5 / 80

Height = Old height in Character units * 11 / 66

iii. In Layout Editor **View--View Options--Rulers**

Set the **Units** *Inches*

Grid Spacing *1*

Number of snap points per Grid Spacing can be depend on the granularity of control required, 4 snap points would ensure good granular control.

Let the **Character Cell Size (points)** be Horizontal 6.5 , Vertical 12.

iv. Set the **Format--Font** to Arial – Regular 8.

v. Additionally the following settings are required at the Concurrent Program Definition,

a. Set Options field to **Version=2.0b**
b. Output format shall be HTML/PDF/Postscript.
c. Use a separate Print Style for bitmap reports (Portrait Bit Map/Landscape Bit Map)

Warning: Exercise Caution using Menu Item **Arrange--Group** , because grouping moves all the objects grouped to the same layer.In the case of repeating frames this can lead to a frequency error. For example, suppose a repeating frame contains a boilerplate object and a field. The boilerplate object is a layer above the field.If you group the repeating frame and the boilerplate object , the repeating frame is moved to the same layer as the boilerplate object. When you run the report , you receive a frequency error because the repeating frame is a layer above its enclosed field. (Repeating frames must be a layer below the objects they contain).

Layout Editor/Frames

Most of the customizations of Character Mode reports to Bitmap reports require painting of rectangular or square objects to group/format objects. In these cases , the following points should be considered,

- Do not substitute *frames* with *rectangles*.
  Use a *frame* to enclose objects*, rectangle* is a boilerplate object and does not enclose the objects. Also *rectangles* are fixed in length and width. Whenever we need to draw a rectangular/square object around the objects that repeat use *frame* and give a line color and line weight of 1pt.
  Use *rectangle* to enclose boilerplate objects that are fixed e.g., Column header.

- Paint the object at the correct place.
  To paint a *frame* or any object in a layout editor that already has objects , the following steps could be of help,
  i. Understand the existing layout structure and zoom in to correct location where you would like to place the new object.
  ii. After making sufficient room to paint the object , paint the object so that the appropriate enclosing object physically encloses it.
  iii. An object that appears to have been enclosed by an enclosing object in a layout editor may not be actually enclosed. This is because , an object is considered to be enclosed by another object only if all of the following are true ;
  a) both objects belong to same region (Body,Margin,Header or Trailer);
  b) the outermost of the two objects is a frame or repeating frame;
  c) the outermost of the two objects is behind the other object;
  d) the innermost of the two objects lies entirely within the borders of the other object.
  An object painted in the layout editor which appears to have been enclosed or enclose objects might satisfy a),b) and d) of above but not c). In order that a newly painted object should enclose other existing objects , the object painted needs to be moved backward till it is behind the enclosed objects. Optionally , all other enclosed objects can be moved forward.

- Use **flex** mode judiciously.
  Often , it might be required to paint a frame in an existing layout editor to enclose only certain objects and to make room for additional objects. In this case the new frame painted needs to be larger than the enclosed objects and the other objects that are not enclosed by the newly painted frame need to be pushed in their path.
  In a scenario like the one described above, first decide on the exact additional space needed to be covered by the new frame apart from enclosing the existing objects,
  Second turn the **flex** mode ON and paint the new frame to the exact size. The decision as to the size of the frame should be correct before painting the frame, because once the non-enclosed objects are pushed in their path using flex mode, cannot be pulled back at a later stage.

Applications User Exits

The user exits available in Oracle Reports are:

- FND SRWINIT
- FND SRWEXIT
- FND FORMAT_CURRENCY
- FND FLEXIDVAL
- FND FLEXSQL

FND SRWINIT / FND SRWEXIT

FND SRWINIT sets your profile option values and allows Oracle Application Object Library user exits to detect that they have been called by an Oracle Reports program. FND SRWINIT also allows your report to use the correct organization automatically. FND SRWEXIT ensures that all the memory allocated for Oracle Application Object Library user exits has been freed up properly.

FND FORMAT_CURRENCY

This user exit formats the currency amount dynamically depending upon the precision of the actual currency value, the standard precision, whether the value is in a mixed currency region, the user's positive and negative format profile options, and the location (country) of the site. The location of the site determines the thousands separator and radix to use when displaying currency values. An additional profile determines whether the thousands separator is displayed. Use the Currencies window to set the standard, extended, and minimum precision of a currency.

You obtain the currency value from the database into an Oracle Reports column. Define another Oracle Reports column, a formula column of type CHAR, which executes the FORMAT_CURRENCY user exit to format the currency value. A displayed field has this formula column as its source so that the formatted value is automatically copied into the field for display.

Syntax

```
FND FORMAT_CURRENCY
CODE=":column containing currency code"
DISPLAY_WIDTH="field width for display"
AMOUNT=":source column name"
DISPLAY=":display column name"
[MINIMUM_PRECISION=":P_MIN_PRECISION"]
[PRECISION="{STANDARD|EXTENDED}"]
[DISPLAY_SCALING_FACTOR="":P_SCALING_FACTOR"]
```

Procedure

Step 1. Define Your Parameters

First define all the parameters (using the Oracle Reports Parameter Screen). Use these parameters in the user exit calls and SQL statements.

```
Name: P_CONC_REQUEST_ID
Data Data Type: NUMBER
Width: 15
Initial Value: 0
```

You always create this lexical parameter. "FND SRWINIT" uses this parameter to retrieve information about this concurrent request.

```
Name: P_MIN_PRECISION
Data Type: NUMBER
Width: 2
Initial Value:
```

You reference this lexical parameter in your FND FORMAT_CURRENCY user exit call.

Step 2. Call FND SRWINIT

You always call FND SRWINIT from the Before Report Trigger as follows:

```
SRW.USER_EXIT('FND SRWINIT');
```

This user exit sets up information for use by profile options and other AOL features.

You always call FND SRWEXIT from the After Report Trigger as follows:

```
SRW.USER_EXIT('FND SRWEXIT');
```

This user exit frees all the memory allocation done in other AOL exits.

Step 3. Create the Currency Code Query

Create a query which selects the currency code and the currency amount from your table. In this case you might use:

```
SELECT OFFICE,
SUM(AMOUNT) C_INCOME,
CURRENCY_CODE C_CURRENCY
FROM OFFICE_INCOME
WHERE TRANSACTION_DATE = '01/92'
ORDER BY BY OFFICE
```

Step 4. Create a column for the currency call

Create one column (C_NET_INCOME) which contains the user exit (FND FORMAT_CURRENCY) call. This is a formula column which formats the number and displays it. The user exit call looks like the following:

```
SRW.REFERENCE(:C_CURRENCY);
SRW.REFERENCE(:C_INCOME);
SRW.USER_EXIT('FND FORMAT_CURRENCY
CODE=":C_CURRENCY"
DISPLAY_WIDTH="15"
AMOUNT=":C_INCOME"
DISPLAY=":C_NET_INCOME"
MINIMUM_PRECISION=":P_MIN_PRECISION"');
RETURN(:C_NET_INCOME);
```

Suggestion: Always reference any source column/parameter which is used as a source for data retrieval in the user exit. This guarantees that this column/parameter will contain the latest value and is achieved by "SRW.REFERENCE" call as shown above.

Here the column name containing currency code is "C_CURRENCY" and the field width of the formatted amount field is 15. The source column is "C_INCOME" and the resulting formatted output is placed in "C_NET_INCOME". The minimum precision of all the currencies used for this report is retrieved from the lexical P_MIN_PRECISION (which in this case is set to 3). At the end of the user exit call remember to reference the column "C_NET_INCOME" by RETURN(:C_NET_INCOME), otherwise the column may not contain the current information. You do not include the MINIMUM_PRECISION token for single currency reports.

Step 5. Hide the Amount

In Default layout, deselect the amount column (C_INCOME) so that it is not displayed in the report. Do not display this amount because it contains the unformatted database column value. In the layout painter update the boiler plate text for each displayed currency field (which in this case are C_CURRENCY and C_NET_INCOME)

Attention: Repeat steps 4 and 5 for each displayed currency field.

Step 6. Create the title

In the layout painter paint the boiler plate text title as follows moving previous fields and boiler plate text as necessary:

```
Net Income for January 1992
```

Step 7. Define Your Report with Application Object Library

Define your report with Standard Request Submission. Ensure you define an argument P_MIN_PRECISION which defaults to $PROFILE$.MIXED_PRECISION.
The report is now ready to be run.

Summary

A brief summary of the report specifications:
Lexical Parameters:

- P_CONC_REQUEST_ID (required)
- P_MIN_PRECISION (needed for mixed currency reports)
- Column Names:
- C_CURRENCY
- C_NET_INCOME
- AOL User Exits:
- FND SRWINIT (required)
- FND FORMAT_CURRENCY
- FND SRWEXIT (required)

Using Flexfield APIs

Using Oracle Applications flexfields routines with Oracle Reports, you can build reports that display flexfields data easily and in a number of ways:

- Display any individual segment value, prompt, or description.
- Display segment values, prompts, or descriptions from multiple flexfield structures (or contexts) in the same report.
- Display segment values, prompts, or descriptions from different flexfields in the same report.
- Display two or more flexfield segment values, prompts, or descriptions, concatenated with delimiters, in the correct order.
- This includes description information for dependent, independent, and table validated segments.
- Restrict output based upon a flexfield range (low and high values).
- Prevent reporting on flexfield segments and values that users do not have access to (flexfield value security).
- Specify order by, group by, and where constraints using one or more, or all segment columns.

General Methodology :

You use a two step method to report on flexfield values. The first step creates the appropriate SQL statement dynamically based upon the user's flexfield. The output of the first step is used as input to the second step. The second step formats this raw data for display.
Step 1 (Construction):

The first step requires you to include one or more lexical parameters (Oracle Reports variables that can be changed at runtime) in your SQL statement. You call the user exit FND FLEXSQL with different arguments to specify that part of the query you would like to build.
The user exit retrieves the appropriate column names (SQL fragment) and inserts it into the lexical parameter at runtime before the SQL query is executed. The query then returns site– and runtime–specific flexfield information. For example, suppose you have the following query:

SELECT &LEXICAL1 alias, column
FROM table
WHERE &LEXICAL2
The preliminary calls to FND FLEXSQL replace values of LEXICAL1 and LEXICAL2 at execution time with the SQL fragments. For example, LEXICAL1 becomes ”SEGMENT1||'\n'||SEGMENT2” and LEXICAL2 becomes ”SEGMENT1 < 2” (assuming the user's flexfield is made up of two segments and the user requested that the segment value of SEGMENT1 be less than 2). The actual executed SQL query might be:
SELECT SEGMENT1||'\n'||SEGMENT2 alias, column
FROM table
WHERE SEGMENT1 < 2
The SQL statement for a user with a different flexfield structure might be:
SELECT SEGMENT5||'\n'||SEGMENT3||'\n'||SEGMENT8 alias, column
FROM table
WHERE SEGMENT3 < 2
With this step you can alter the SELECT, ORDER BY, GROUP BY, or WHERE clause. You use this step to retrieve all the concatenated flexfield segment values to use as input to the user exit FND FLEXIDVAL in step 2 (described below). You call this user exit once for each lexical parameter you use, and you always call it at least once to get all segments. This raw flexfield information is in an internal format and should never be displayed (especially if the segment uses a ”hidden ID” value set).

Step 2 (Display):

The second step requires you to call another user exit, FND FLEXIDVAL, on a ”post–record” basis. You create a new formula column to contain the flexfield information and include the user exit call in this column. This user exit determines the exact information required for display and populates the column appropriately. By using the flexfield routines the user exit can access any flexfield information. Use this step for getting descriptions, prompts, or values. This step derives the flexfield information from the already selected concatenated values and populates the formula column on a row by row basis. You call FND FLEXIDVAL once for each record of flexfield segments. The flexfield user exits for Oracle Reports are similar to their Oracle Application Object Library (using SQL*Forms) counterparts LOADID(R) or LOADDESC and POPID(R) or POPDESC; one to construct or load the values (FLEXSQL), the other to display them (FLEXIDVAL). The token names and meanings are similar.

Basic Implementation Steps

Step 1 Call FND SRWINIT from your Before Report Trigger
You call the user exit FND SRWINIT from your Before Report Trigger. FND SRWINIT fetches concurrent request information and sets up profile options. You must include this step if you use any Oracle Application Object Library features in your report (such as concurrent processing).

Step 2 Call FND SRWEXIT from your After Report Trigger

You call the user exit FND SRWEXIT from your After Report Trigger. FND SRWEXIT frees all the memory allocation done in other Oracle Applications user exits. You must include this step if you use any Oracle Application Object Library features in your report (such as concurrent processing).

Step 3 Call FND FLEXSQL from the Before Report Trigger

You need to pass the concatenated segment values from the underlying code combinations table to the user exit so that it can display appropriate data and derive any descriptions and values from switched value sets as needed. You get this information by calling the AOL user exit FND FLEXSQL from the Before Report Trigger. This user exit populates the lexical parameter that you specify with the appropriate column names/SQL

fragment at run time. You include this lexical parameter in the SELECT clause of your report query. This enables the report itself to retrieve the concatenated flexfield segment values. You call this user exit once for each lexical to be set. You do not display this column in your report. You use this ”hidden field” as input to the FND FLEXIDVAL user exit call. This user exit can also handle multi–structure flexfield reporting by generating a decode on the structure column. If your report query uses table joins, this user exit can prepend your code combination table name alias to the column names it returns.
SELECT &LEXICAL alias, column
becomes, for example,
SELECT SEGMENT1||'\n'||SEGMENT2 alias, column
Note: Oracle Reports needs the column alias to keep the name of column fixed for the lexicals in SELECT clauses. Without the alias, Oracle Reports assigns the name of the column as the initial value of the lexical and a discrepancy occurs when the value of the lexical changes at run time.

Step 4 Restrict report data based upon flexfield values

You call the user exit FND FLEXSQL with MODE=”WHERE” from the Before Report Trigger. This user exit populates a lexical parameter that you specify with the appropriate SQL fragment at run time. You include this lexical parameter in the WHERE clause of your report query. You call this user exit once for each lexical to be changed. If your report query uses table joins, you can have this user exit prepend your code combination table name alias to the column names it returns.
WHERE tax_flag = 'Y' and &LEXICAL < &reportinput
becomes, for example,
WHERE tax_flag = 'Y' and T1.segment3 < 200
The same procedure can be applied for a HAVING clause.

Step 5 Order by flexfield columns

You call the user exit FND FLEXSQL with MODE=”ORDER BY” from the Before Report Trigger. This user exit populates the lexical parameter that you specify with the appropriate SQL fragment at run time. You include this lexical parameter in the ORDER BY clause of your report query. You call this user exit once for each lexical to be changed. If your report query uses table joins, you can have this user exit prepend your code combination table name alias to the column names it returns.
ORDER BY column1, &LEXICAL
becomes, for example,
ORDER BY column1, segment1, segment3

Step 6 Display flexfield segment values, descriptions, and prompts

Create a Formula Column (an Oracle Reports 2.5 data construct that enables you to call a user exit). Call the user exit FND FLEXIDVAL as the Formula for this column. This user exit automatically fetches more complicated information such as descriptions and prompts so that you do not have to use complicated table joins to the flexfield tables. Then you create a new field (an Oracle Reports 2.5 construct used to format and display Columns), assign the Formula Column as its source, and add this field to your report using the screen painter. You need to include this field on the same Repeating Frame (an Oracle Reports 2.5 construct found in the screen painter that defines the frequency of data retrieved) as the rest of your data, where data could be actual report data, boilerplate, column headings, etc. The user exit is called and flexfield information retrieved at the frequency of the Repeating Frame that contains your field. In the report data case, the user exit is called and flexfield information retrieved once for every row retrieved with your query. All flexfield segment values and descriptions are displayed left justified. Segment values are not truncated, that is, the Display Size defined in Define Key Segments screen is ignored. Segment value descriptions are truncated to the description size (if one is displayed) or the concatenated description size (for concatenated segments) defined in the form.

FND FLEXSQL

Call this user exit to create a SQL fragment usable by your report to tailor your SELECT statement that retrieves flexfield values. This fragment allows you to SELECT flexfield values or to create a WHERE, ORDER BY, GROUP BY, or HAVING clause to limit or sort the flexfield values returned by your SELECT statement. You call this user exit once for each fragment you need for your select statement. You define all flexfield columns in your report as type CHARACTER even though your table may use NUMBER or DATE or some other datatype.

Syntax:

```
FND FLEXSQL
CODE=" flexfield code"
APPL_SHORT_NAME=" application short name"
OUTPUT=": output lexical parameter name"
MODE="{ SELECT | WHERE | HAVING | ORDER BY}"
[DISPLAY="{ALL | flexfield qualifier | segment number}"]
[SHOWDEPSEG="{Y | N}"]
[NUM=": structure defining lexical" |
MULTINUM="{Y | N}"]
[TABLEALIAS=" code combination table alias"]
[OPERATOR="{ = | < | > | <= | >= | != | "||"|
BETWEEN | QBE}"]
[OPERAND1=": input parameter or value"]
[OPERAND2=": input parameter or value"]
```

FND FLEXIDVAL

Call this user exit to populate fields for display. You pass the key flexfields data retrieved by the query into this exit from the formula column. With this exit you display values, descriptions and prompts by passing appropriate token (any one of VALUE, DESCRIPTION, APROMPT or LPROMPT).

Syntax:

```
FND FLEXIDVAL
CODE=" flexfield code"
APPL_SHORT_NAME=" application short name"
DATA=": source column name"
[NUM=": structure defining source column/lexical"]
[DISPLAY="{ALL| flexfield qualifier| segment number}"]
[IDISPLAY="{ALL| flexfield qualifier| segmentnumber}"]
[SHOWDEPSEG="{Y | N}"]
[VALUE=": output column name"]
[DESCRIPTION=": output column name"]
[APROMPT=": output column name"]
[LPROMPT=": output column name"]
[PADDED_VALUE=": output column name"]
[SECURITY=": column name"]
```

Flexfields Report–Writing Steps

These are the basic steps you use every time you write an Oracle Reports report that accesses flexfields data. This section assumes you already have a thorough knowledge of Oracle Reports. Though these examples contain only the Accounting Flexfield, you can use these methods for any key flexfield.

Step 1 Define your Before Report Trigger
(this step is always the same)

You always call FND SRWINIT from the Before Report Trigger:

```
SRW.USER_EXIT('FND SRWINIT');
```

This user exit sets up information for use by flexfields, user profiles, the concurrent manager, and other Oracle Applications features. You must include this step if you use

any Oracle Application Object Library features in your report (such as concurrent processing).

Step 2 Define your After Report Trigger
(this step is always the same)

You always call FND SRWEXIT from the After Report Trigger:
SRW.USER_EXIT('FND SRWEXIT');

This user exit frees all the memory allocation done in other Oracle Applications user exits. You must include this step if you use any Oracle Application Object Library features in your report (such as concurrent processing).

Step 3 Define your required parameters

You define the parameters your report needs by using the Data Model Painter. You use these parameters in the user exit calls and SQL statements.

Lexical Parameters				
Name	Data Type	Width	Initial Value	Notes
P_CONC_REQUEST_ID	Number	15	0	Always create
P_FLEXDATA	Character	approximately 600 (single structure) to 6000 (roughly ten structures)	Long string	Cumulative width more than expected width required to hold data

You must always create the P_CONC_REQUEST_ID lexical parameter. "FND SRWINIT" uses this parameter to retrieve information about the concurrent request that started this report. The P_FLEXDATA parameter holds the SELECT fragment of the SQL query. The initial value is used to check the validity of a query containing this parameter and to determine the width of the column as specified by the column alias. Its initial value is some string that contains columns with a cumulative width more than the expected width required to hold the data. Make sure the width of this column is sufficient. If there are total 30 segments in the table then the safest initial value will be: (SEGMENT1||'\n'||SEGMENT2||'\n'||SEGMENT3 ... SEGMENT30)
You determine the width by determining the length of that string. That length is roughly the number of characters in the table alias plus the length of the column name, times the number of segments your code combinations table contains, times the number of structures you expect, plus more for delimiter characters as shown in the string above.

Step 4 Define your other parameters

You define the rest of the parameters your report needs by using the Data Model Painter. You use these parameters in the user exit calls and SQL statements.

Step 5 Call FND FLEXSQL from your Before Report Trigger to populate P_FLEXDATA

Next, given that you want to display flexfield information like concatenated values and descriptions, and arrange them in order, you make one call to FND FLEXSQL from the Before Report Trigger specifying the lexical parameters. This call changes the value of the lexical parameter P_FLEXDATA at runtime to the SQL fragment that selects all flexfields value data. For example, the parameter changes to (SEGMENT1||'\n'||SEGMENT2||'\n'||SEGMENT3||'\n'||SEGM ENT4).

When you incorporate this lexical parameter into the SELECT clause of a query, it enables the query to return the concatenated segment values that are needed as input to other AOL user exits. These exits then retrieve the actual flexfield information for display purposes.

Here is an example FND FLEXSQL call. Notice that the arguments are very similar to other flexfield routine calls; CODE= and NUM= designate the key flexfield and its structure, respectively. For a report on a different key flexfield (such as the System Items flexfield), you would use a different CODE and NUM.

```
SRW.REFERENCE(:P_STRUCT_NUM);
SRW.USER_EXIT('FND FLEXSQL
CODE="GL#"
NUM=":P_STRUCT_NUM"
APPL_SHORT_NAME="SQLGL"
OUTPUT=":P_FLEXDATA"
MODE="SELECT"
DISPLAY="ALL"');
```

You should always reference any source column/parameter that is used as a source for data retrieval in the user exit. This guarantees that this column/parameter will contain the latest value and is achieved by "SRW.REFERENCE" call as shown above.

Step 6 Call FND FLEXSQL from your Before Report Trigger to populate other parameters

You call FND FLEXSQL once for every lexical parameter such as P_WHERE or P_ORDERBY.

Step 7 Define your report query or queries

Define your report query Q_1:

```
SELECT &P_FLEXDATA C_FLEXDATA
FROM CODE_COMBINATIONS_TABLE
WHERE CODE_COMBINATIONS_TABLE.STRUCTURE_DEFINING_COLUMN
        = &P_STRUCT_NUM
```

The query fetches the data required to be used as input for the FLEXIDVAL user exit later.

Note: Always provide a column alias (C_FLEXDATA in this example) in the SELECT clause that is the name of column. This name of the column is required in FND FLEXIDVAL.
When the report runs, the call to FND FLEXSQL fills in the lexical parameters. As a result the second query would look something like:

```
SELECT (SEGMENT1||'–'||SEGMENT2||'–'||SEGMENT3||'–'||
SEGMENT4) C_FLEXDATA
FROM CODE_COMBINATIONS_TABLE
WHERE
CODE_COMBINATIONS_TABLE.STRUCTURE_DEFINING_COLUMN =
101
```

Step 8 Create formula columns

Now create columns C_FLEXFIELD and C_DESC_ALL (and any others your report uses) corresponding to the values and descriptions displayed in the report. They all are in group G_1. Be sure to adjust the column width as appropriate for the value the column holds (such as a prompt, which might be as long as 30 characters).

Step 9 Populate segment values formula column

To retrieve the concatenated flexfield segment values and description, you incorporate the flexfields user exits in these columns. In the column definition of C_FLEXFIELD, you incorporate the FND FLEXIDVAL user exit call in the formula field. You pass the concatenated segments along with other information to the user exit, and the user exit populates the concatenated values in this column as specified by the VALUE token.
A typical call to populate segment values in this column looks as follows:

```
SRW.REFERENCE(:P_STRUCT_NUM);
SRW.REFERENCE(:C_FLEXDATA);
SRW.USER_EXIT('FND FLEXIDVAL
CODE="GL#"
NUM=":P_STRUCT_NUM"
APPL_SHORT_NAME="SQLGL"
DATA=":C_FLEXDATA"
VALUE=":C_FLEXFIELD"
DISPLAY="ALL"');
RETURN(:C_FLEXFIELD);
```

Step 10 Populate segment descriptions

To populate the segment description use
DESCRIPTION="C_DESC_ALL" instead of VALUE="C_FLEXFIELD"
as in the previous call. The user exit call becomes:

```
SRW.REFERENCE(:P_STRUCT_NUM);
SRW.REFERENCE(:C_FLEXDATA);
SRW.USER_EXIT('FND FLEXIDVAL
CODE="GL#"
NUM=":P_STRUCT_NUM"
APPL_SHORT_NAME="SQLGL"
DATA=":C_FLEXDATA"
DESCRIPTION=":C_DESC_ALL"
DISPLAY="ALL"');
RETURN(:C_DESC_ALL);
```

You have created parameters and columns that are containers of all the values to be displayed. Now, in the following steps, you create the layout to display these values on the report.

Step 11 Create your default report layout

First choose Default Layout to generate the default layout. Deselect C_FLEXDATA. Specify a "Label" and a reasonable "Width" for the columns you want to display.

Default Layout Column Settings

Column	Label	Width
C_FLEXFIELD	Accounting Flexfield	30
C_DESC_ALL	Flexfield Description	50

Oracle Reports takes you to the layout painter. Generate and run the report.

Step 12 Finish your report

Adjust your report layout as needed.

☞	**Attention:** For obtaining Oracle Applications information use PL/SQL APIs , wherever available , instead of user exits. Because Oracle may not support the user exits in future releases. For example, use FND_PROFILE.GET to get the profile values instead of user exit FND GETPROFILE.

Debugging techniques

Use PL/SQL exception section to trap error conditions in the report execution. Use SRW.MESSAGE to trap the location of error. In case of errors like “Object not fully enclosed “ refer to the Layout Editor/Frames section of this document for conditions of enclosure of objects, and accordingly rectify the objects in the Layout Editor.

Variable naming and usage

Variables in PL/SQL program units shall following the naming conventions as laid out in the Technical Design Standards document (*Dev_Std.doc*).

No hardcoding of any values shall be done, instead , use of declared constants shall be made ;and if any constants are declared it should be documented specifying the purpose.

Performance improvement techniques

An Oracle Applications Report’s performance mainly depends on the SQL’s performance, as the issue of network traffic is nonexistent.

The following guidelines shall be followed for SQL tuning.

Before approaching to optimize an SQL it is imperative to have an understanding of the size of the tables involved. In a development environment , where the development is taking place on demo/test database it would not be possible to have proper size of data to test an SQL’s performance. In an Oracle Applications development scenario , a developer can estimate a table’s relative size depending on , the client’s business scenario , and the functionality of the table. Based on these factors , the developer should arrive at relative sizes of tables for SQL performance tuning.

1.Use the table that returns the least number of rows as the driving table.
A driving table is one that drives the query , i.e., the rows of the driving table will be used to evaluate other conditions.
Most of the times Oracle takes the last table in the FROM clause as driving table. In case last table of the FROM clause is not taken as driving , then , a driving table can be forced using hint USE_NL (table_name).

2. Use IN and EXISTS operators judiciously.
Developers should be aware that, the efficiency of EXISTS and IN is dependant on the amount of data in each table. A query with IN in it drives from the subquery accessing the main query for each row returned, when, a query uses EXISTS it drives from the main query accessing the subquery for each row returned. So if the subquery returns few rows, but, the main query returns a lot of rows for each row from the subquery use the IN operator , the opposite would be the case for EXISTS operator.

3. Accidental disabling of Indexes.
   If a function , whether explicit or implicit, is used on an indexed column in a WHERE clause, the index would not be used. So a developer should keep in mind to avoid accidental disabling of indexes. Similarly , this case would also be helpful to disable an index intentionally. It would be advisable to disable an index when the data retrieved from the table would be more than 20% of the total data.

4. Avoid using NOT IN operator.
   Use of NOT IN operator disables index usage. This is because Oracle assumes few records satisfy the condition and hence performs a FTS (Full Table Scan). Queries can be reworded to avoid NOT IN operator. The following use of an outer join is a good alternative.
   For e.g., The query selects all departments with no employees,

```
SELECT deptno,dname
FROM dept
WHERE deptno NOT IN (SELECT emp.deptno FROM emp);
```

   Instead use,

```
SELECT a.deptno,a.dname
FROM  dept a,emp b
WHERE a.deptno = b.deptno (+)
AND b.rowid is null;
```

5. Use EXPLAIN PLAN / SQL Trace.
   Always use EXPLAIN PLAN to identify the SQL execution path and accordingly tune the statement.

   The following Query may be used to retrieve the execution plan from PLAN TABLE.

```
COL Id FOR a3 TRU
COL  Parent_id FOR a6 TRU
COL Operation FOR a35 TRU
COL Option FOR a13 TRU
COL Object FOR a10 TRU
COL Object_Type FOR a12 TRU

SELECT      id,
            parent_id,
            LPAD(' ',2*LEVEL)||OPERATION Operation ,
            OPTIONS Option,
            OBJECT_NAME Object,
            OBJECT_TYPE Object_Type
FROM        PLAN_TABLE
WHERE       STATEMENT_ID='<statement_id>'
CONNECT BY PRIOR ID = PARENT_ID
AND          STATEMENT_ID = '<statement_id>'
START WITH ID = 0
AND          STATEMENT_ID = '<statement_id>'
ORDER BY ID
```

6. Use minimum number of queries while designing a report using Reports 2.5 tool.
   This will reduce the parse time of Reports 2.5 engine. Also , use of an SQL function is advised instead of a Reports 2.5 summary column , wherever feasible.

Exception handling

Always handle all the exception conditions using the PL/SQL Exceptions. Meaningful user defined exceptions should also be used to trap specific functional exceptions and provide an action.

Use the Reports 2.5 built in exception SRW.USER_EXIT_FAILURE to check the failure of any user exit called via SRW.USER_EXIT. The exception section should trap the user exit name that has caused the exception to raise. This can be achieved by defining a variable and setting its value after the calls to the user exits.

Error messages

All the user defined error messages in Reports should , along with the error message , specify an action. The error numbers used should be within the specific range of error numbers assigned for each work unit.

Porting considerations

All reports developed for Oracle Applications Release 11 shall be developed using Developer 2000 Release 1.6.1. The PL/SQL version supported by Reports 2.5 is 1.2 and does not support certain features of PL/SQL Version 2.0 upwards. Hence any stored program units that are called in Reports PL/SQL should not use the features not supported by PL/SQL Version 1.2. Some of this exclusive features are PL/SQL tables/records,Dynamic SQL.

In case of attached libraries path should be removed at the time of generation of Report module.

Source Code Control

Source code control is the process of managing revisions to program source code and the modification of files by multiple individuals. Microsoft Visual SourceSafe v 5.0 (VSS) should be used to control versions of Reports. In case of customization of report the first version would always be the original report as provided by Oracle Applications.

Using VSS

Microsoft Visual SourceSafe 5.0 is a project-oriented version control system for all types of files, including text files, graphics files, binary files, sound files, and video files. Using Visual SourceSafe, you can track changes made to a file from the moment it was created. And you can merge changes from two or more different versions of a file into one file that contains them all.

Working With Files

When you want to modify a file, you check it out of the database. Visual SourceSafe copies the file from the database into your working folder. You can then edit the file. If anyone else attempts to check out the same file for editing, Visual SourceSafe generates a message stating the file is already checked out. This simple checkout protocol ensures that conflicts do not arise among multiple users working on the same file.

After you are done editing the file, you check it into Visual SourceSafe using the Check In command. This copies the modified file from your folder into the Visual SourceSafe's database, making your changes accessible to other users. However, Visual SourceSafe stores all the changes that have been made to the file the most recent copy is always available, but earlier versions can be retrieved as well. Visual SourceSafe's *reverse delta* technology ensures that all versions of a file are available, but uses a minimum of disk space.

If you want to check something in a file, but don't need to edit it, you can use the Get Latest Version command to get the most recent version of the file into your working folder. You can use SourceSafe's Show History command to conveniently view the file or project's history, and the Show Differences command to determine differences between a file in your local folder and the latest version of that file stored in the SourceSafe database.

By far, the most commonly used Visual SourceSafe commands are those that copy files into and out of the Visual SourceSafe database during day-to-day use.

Getting Files

When you want access to a file for viewing or compiling, but not for modification, use the Get Latest Version command. This copies the file from the current project into your working folder. The file Visual SourceSafe creates is read-only any modifications cannot be saved.

To get the most recent version of a file :

- Click the file you want in Visual SourceSafe Explorer, and on the SourceSafe menu, click Get Latest Version.

To get an earlier version of a file :

1. Click the file you want in Visual SourceSafe Explorer, and on the Tools menu, click Show History.

2. In the History of File dialog box, click the version of the file you want, and then click Get Latest Version.

Viewing Files

You can view the master copy of a file without placing a local copy of the file in your working folder.

To view a file

- Click the file you want in Visual SourceSafe Explorer, and on the Edit menu, click View File.

Editing Files

You can edit a file in your working folder by double-clicking the file in the file list. In the confirmation dialog, choose Checkout this file, and edit it in your working folder. Visual SourceSafe ensures that the file is checked out, and opens it in the editor associated with the file extension.

Checking Out and Checking In Files

To edit a file, you must check it out of the Visual SourceSafe database. The Check Out command creates a writable copy of the file from the project in your working folder. A file check out is generally exclusive, that is, no one else can check out a file that you have checked out. Visual SourceSafe indicates who has a file checked out in the User column of the file pane.

You can complete your check out operation in one of two ways. You can check your updated file into Visual SourceSafe, storing your changes in the current project. Or, you can undo your check out, which cancels your changes, both in Visual SourceSafe and in your working folder the file returns to the way it was before you checked it out.

To check out a file

- Click the file you want in Visual SourceSafe Explorer, and on the SourceSafe menu, click Check Out.

To check in a file, saving your changes

- Click the file you want in Visual SourceSafe Explorer, and on the SourceSafe menu, click Check In.

To check in a file, undoing your changes

- Click the file you want in Visual SourceSafe Explorer, and on the SourceSafe menu, click Undo Check Out.

Warning : If you choose the Undo Check Out command, you will lose any changes you have made to the local copy of your file(s).

Open and Closed Issues

Open Issues

Closed Issues