Dubugging HRMS Fast Formula

An Oracle White Paper
May 2005

EXECUTIVE OVERVIEW

This paper is produced in response to a request from Oracle Support for additional information about how to diagnose issues in Fast Formula. The paper is suitable for both internal and external use.

It includes a number of techniques for debugging formulae. It also discusses some of the common errors that arise when writing/compiling formulae and provides suggested approaches for correcting them.

INTRODUCTION

Oracle Fast Formula is used widely in the HRMS product to provide developers and implementers the capability to configure validation and processing rules and calculations. It uses a simple programming language that recognizes objects that exist in the HRMS schema, allowing the formula to process HRMS data items directly.

When writing formula it is useful to be able to determine the processing flow so that diagnosis or debugging of problems can be performed easily. In the Debugging Techniques section this paper discusses 3 methods for extracting logging/debugging information, these include using the standard output form the formula, using the native fast formula engine logging output and embedded trace statements in the formula itself.

There are a number of common errors that can arise when either compiling or running formulae. Techniques for identifying the causes of these errors and suggested fixes are discussed in the Fast Formula Errors section.

The paper is intended for anyone who is developing, supporting or implementing fast formulae. It does not contain any HRMS product specific information and it does not discuss best practice techniques for implementing functionality using Fast Formulae.

There is a list of related documents that are available on Metalink in Appendix E.

DEBUGGING TECHNIQUES

There are essentially 3 methods for extracting logging or debug output from a Fast Formula. Theses are as follows:

1. Using the Message output from the formula
2. Using the native Fast Formula engine logging capabilities
3. Embedding trace statements in the formula and using PYUPIP to capture the output.

These 3 techniques are discussed below.

Message output from Fast Formulae

The simplest method of passing debug messages out of a fast formula is to use the Fast Formula RETURN statement. The value from the RETURN statement will appear in the Run Results form if the payroll is being run through payroll, otherwise it will appear in the outputs of the formula when processed in PL/SQL.

The following example will output a value in the MESG return variable:

```
/*
   Formula Name : TEST
   Description  : Test formula
   Contexts     : DATE_EARNED, ASSIGNMENT_ID
*/

default for per_first_name is ' '
default for per_last_name is ' '

mesg = 'Person is ' + per_first_name + ' ' + per_last_name

return mesg
```

Although this techniques is quick and easy to use it is limited in use because the message variable can only ever have 1 value. If additional trace output were required further messages would have to be added to the formula. This limitation can be overcome using the PYUPIP method - see below.

Logging from Fast Formula execution engine

The Fast Formula execution engine is the PL/SQL or C code that actually executes the formulae when initiated from a calling HRMS application.

It can be configured to provide some useful debugging information including

- Formula definition and corresponding PL/SQL wrapper package.
- The values of the formula inputs and outputs.
- The database items used in the formula and their derived values.

The logging information is sent to either a PYUPIP trace or in the case of the payroll run to the log file (if the LOGGING PAY_ACTION_PARAMETER value is set).

To enable the Fast Formula logging feature a set of switches are used, each switch controls a different portion of the logging output allowing you to control the level of output you receive.

There are 2 ways to enable the Fast Formula engine debug, the first is to enable the switches in PL/SQL code, and the second is to use a profile option.

Enabling the Fast Formula Logging through code

The code method uses has the following syntax:

```
ff_utils.set_debug({ff switch 1} + {ff switch 2} . . .);
```

where the ff switches are listed in the table below.

Enabling the Fast Formula Logging through a profile

The alternate method uses the HR: FastFormula debug level (FF_DEBUG) profile option. Its value is set to a string of characters each of which represents one of the desired switches. The characters for each setting are also shown in the table below. (N.B. When setting this profile its value should always start with 'X' to indicate that 'user exit' logging should be allowed.).

Fast Formula Logging Switches

The following table lists the set of Fast Formula logging switches and indicates the type of output generated by each switch:

Switch	Profile	PL/SQL	Trace Output
Routing Information	R	ff_utils.ROUTING	This outputs information on the procedures accessed during execution. The trace will output: In: <procedure_name> Out <procedure_name>
General Debug	F	ff_exec.FF_DBG	Covers any general debug information relevant to the module, not for a specific formula.
General FF Cache Information	C	ff_exec.FF_CACHE_DBG	This outputs information held in the formula cache for a specific formula. This includes the fdiu rows, but not the MRU chain.
Database Item Cache Debug	D	ff_exec.DBI_CACHE_DBG	Gives information about the database item cache entries.
MRU/LRU Chain Debug	M	ff_exec.MRU_DBG	Gives information about the MRU chain entries.
User Input and Output Table Debug	I	ff_exec.IO_TABLE_DBG	User input and output table debug. Gives information about the input and output tables that are accessed by the user to communicate with formula.

The following examples show how to enable the FF engine logging for Routing and General Cache information.

Explicit setting in PL/SQL:

```
ff_utils.set_debug(ff_utils.ROUTING
                  + ff_exec.FF_CACHE_DEBUG);
```

Via a profile option:

```
Fnd_profile.put('FF_DEBUG','XRC');
```

Interpreting the output

For the example formula above (in the listed Message Output from Fast Formula section) the following output is piped into the PYUPIP trace when it is run using the PL/SQL test function.

Routing Information (profile setting = R)

```
In  : load_formula
In  : ff_fetch
In  : write_dbi_cache
In  : find_dbi_cache_entry
Out : find_dbi_cache_entry
Out : write_dbi_cache
In  : write_dbi_cache
```

General Debug Information (profile setting = F)

No additional logging information

General FF Cache Information (profile setting = C)

```
<- Free Chunk List ->
ff_fetch add to FDIU from end of list:1,4

FMLA CACHE info for formula_id 63208
--------------------------------------
Eff Start   : 16-MAY-2005
Eff End     : 31-DEC-4712
Fmla Name   : JRTEST2
Package     : FFP63208_16052005
First FDIU  : 1
FDIU Count  : 4
Ctx count   : 2
In count    : 0
Out count   : 0

FDIU ROWS
[FDIU]Item Name        V[POS]     I[POS]     Dtype  U  CSum  ContextId
---------------------- ---------- ---------- ------ - ----- ----------
ASSIGNMENT_ID          1          -1         NUMBER U     8        113
DATE_EARNED            1          -1         DATE   U    32        115
PER_FIRST_NAME         1          1          TEXT   D    40    3665134
PER_LAST_NAME          2          2          TEXT   D    40    3665050
```

Useful information includes the formula name and ID, as well as the contexts that will be needed for the formula to execute (the contexts have 'U' in the U column, 'D' represents database items).

Database Item Cache Information (profile setting = D)

```
[DBI]Indx DbItem/Context Name Context Lev Value
--------- ------------------- ----------- ------------------------------
      113 ASSIGNMENT_ID                 8 12167
      115 DATE_EARNED                  32 2005/05/18 05:02:00
        1 PER_FIRST_NAME               40 John-1
       65 PER_LAST_NAME                40 Test-1
[HSH]Indx     First     Count
--------- --------- ---------
  3665050        65         1
  3665134         1         1
.
```

```
.
.
CTXCH: DATE_EARNED          32 2005/05/18 05:02:00 2005/05/18 05:02:12
INVAL: PER_FIRST_NAME       40         32 John
INVAL: PER_LAST_NAME        40         32 Test
```

Useful information includes the actual values passed into the formula as well as the derived values of the DB Items.

MRU/LRU Chain Information (profile setting = M)

No additional logging information

User Input and Output Table Information (profile setting = I)

```
[IT]Index Input/Context Name    Dtype   Class   Value
--------- --------------------- ------- ------- -----------------------
        1 ASSIGNMENT_ID         NUMBER  CONTEXT 12167
        2 DATE_EARNED           DATE    CONTEXT 2005/05/18 05:06:05
```

Useful information includes the values passed in and out of the formula.

Of the above settings the ones that provide the most useful initial logging information are CDI. It is therefore recommended that when setting the HR:FastFormula debug level profile option it should be set to 'XCDI'.

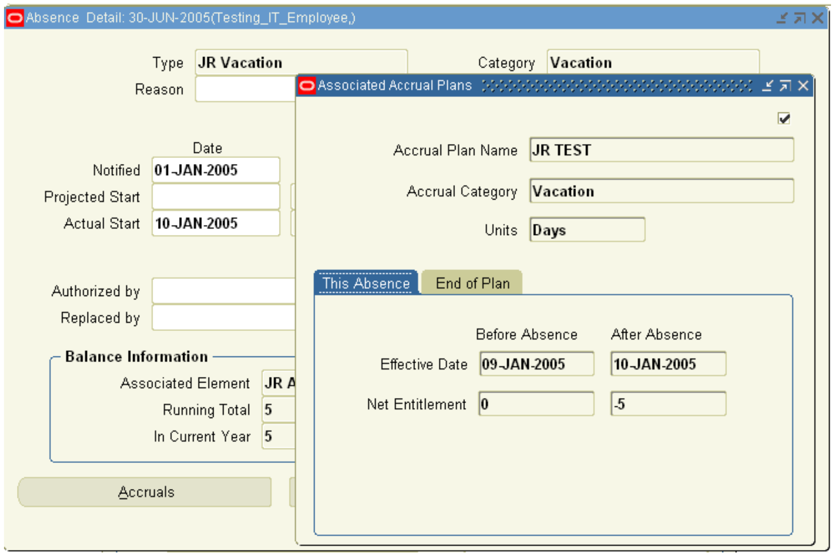
Example: Using seeded PTO formula to derive net accruals.

With the above profile settings in place an accrual plan is created using the seeded PTO_SIMPLE_MULTIPLIER formula that has the following definition:

```
/* -------------------------------------------------------------------
    NAME : PTO_SIMPLE_CARRYOVER
    This formula is the seeded carryover folmula for our simple
    multiplier accrual plan
   -------------------------------------------------------------------
*/

DEFAULT FOR ACP_CONTINUOUS_SERVICE_DATE IS '4712/12/31 00:00:00' (date)
DEFAULT FOR ACP_SERVICE_START_DATE IS '4712/12/31 00:00:00' (date)

INPUTS ARE
Calculation_Date (date),
Accrual_term (text)

Effective_Date = to_date('3105'
                  + to_char(Calculation_date, 'YYYY'), 'DDMMYYYY')

IF  (Accrual_Term = 'CURRENT')
AND (Effective_Date < Calculation_Date) THEN
(
  Effective_date = ADD_YEARS(Effective_Date, 1)
)
ELSE IF (Accrual_term = 'PREVIOUS')
    AND (Effective_Date >= Calculation_Date) THEN
(
  Effective_date = ADD_YEARS(Effective_Date, -1)
)

Expiry_Date = add_years(effective_date, 1)

Max_carryover = 5
Process = 'YES'
```

```
RETURN Max_Carryover, Effective_date, Expiry_Date, Process
```

The calculation of the accrual occurs when querying up the accrual plan in the absence form.



Absence Detail: 30-JUN-2005(Testing_IT_Employee,)

Type	JR Vacation
Category	Vacation
Reason	
Notified	01-JAN-2005
Projected Start	
Actual Start	10-JAN-2005
Authorized by	
Replaced by	

Balance Information

Associated Element	JR A
Running Total	5
In Current Year	5

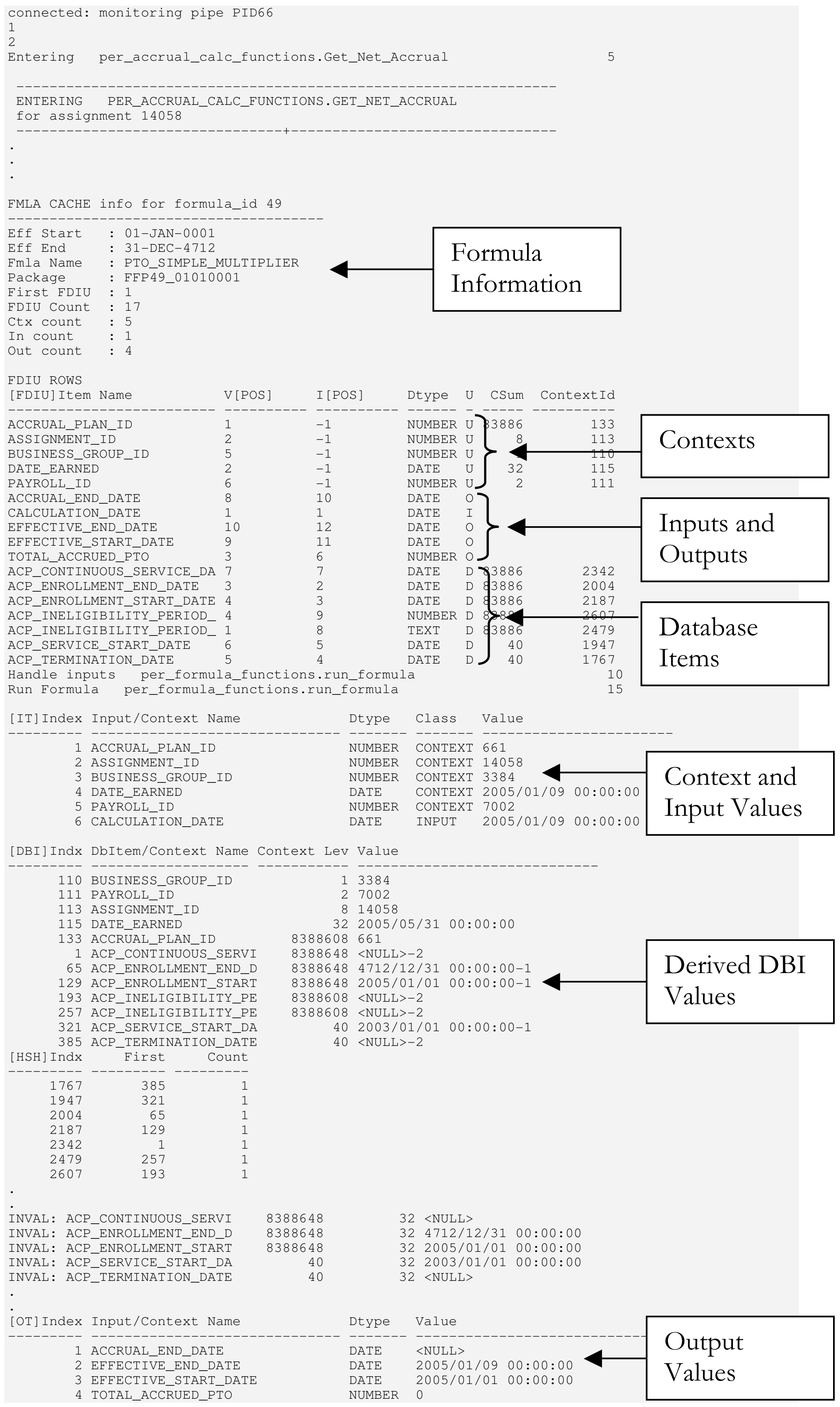
Associated Accrual Plans

Accrual Plan Name	JR TEST
Accrual Category	Vacation
Units	Days

This Absence | End of Plan

	Before Absence	After Absence
Effective Date	09-JAN-2005	10-JAN-2005
Net Entitlement	0	-5

Performing this operation produces the following trace output (FF engine logging information highlighted).



```
connected: monitoring pipe PID66
1
2
Entering   per_accrual_calc_functions.Get_Net_Accrual                   5

 ------------------------------------------------------------------
 ENTERING   PER_ACCRUAL_CALC_FUNCTIONS.GET_NET_ACCRUAL
 for assignment 14058
 ---------------------------------+--------------------------------
.
.
.

FMLA CACHE info for formula_id 49
--------------------------------------
Eff Start   : 01-JAN-0001
Eff End     : 31-DEC-4712
Fmla Name   : PTO_SIMPLE_MULTIPLIER
Package     : FFP49_01010001
First FDIU  : 1
FDIU Count  : 17
Ctx count   : 5
In count    : 1
Out count   : 4

FDIU ROWS
[FDIU]Item Name           V[POS]     I[POS]     Dtype  U  CSum  ContextId
------------------------- ---------- ---------- ------ - ----- ----------
ACCRUAL_PLAN_ID           1          -1         NUMBER U 83886        133
ASSIGNMENT_ID             2          -1         NUMBER U     8        113
BUSINESS_GROUP_ID         5          -1         NUMBER U     1        110
DATE_EARNED               2          -1         DATE   U    32        115
PAYROLL_ID                6          -1         NUMBER U     2        111
ACCRUAL_END_DATE          8          10         DATE   O
CALCULATION_DATE          1          1          DATE   I
EFFECTIVE_END_DATE        10         12         DATE   O
EFFECTIVE_START_DATE      9          11         DATE   O
TOTAL_ACCRUED_PTO         3          6          NUMBER O
ACP_CONTINUOUS_SERVICE_DA 7          7          DATE   D 83886       2342
ACP_ENROLLMENT_END_DATE   3          2          DATE   D 83886       2004
ACP_ENROLLMENT_START_DATE 4          3          DATE   D 83886       2187
ACP_INELIGIBILITY_PERIOD_ 4          9          NUMBER D 83886       2607
ACP_INELIGIBILITY_PERIOD_ 1          8          TEXT   D 83886       2479
ACP_SERVICE_START_DATE    6          5          DATE   D    40       1947
ACP_TERMINATION_DATE      5          4          DATE   D    40       1767
Handle inputs   per_formula_functions.run_formula                        10
Run Formula   per_formula_functions.run_formula                          15

[IT]Index Input/Context Name             Dtype   Class   Value
--------- ------------------------------ ------- ------- -----------------------
        1 ACCRUAL_PLAN_ID                NUMBER  CONTEXT 661
        2 ASSIGNMENT_ID                  NUMBER  CONTEXT 14058
        3 BUSINESS_GROUP_ID              NUMBER  CONTEXT 3384
        4 DATE_EARNED                    DATE    CONTEXT 2005/01/09 00:00:00
        5 PAYROLL_ID                     NUMBER  CONTEXT 7002
        6 CALCULATION_DATE               DATE    INPUT   2005/01/09 00:00:00

[DBI]Indx DbItem/Context Name Context Lev Value
--------- ------------------- ----------- ------------------------------
      110 BUSINESS_GROUP_ID             1 3384
      111 PAYROLL_ID                    2 7002
      113 ASSIGNMENT_ID                 8 14058
      115 DATE_EARNED                  32 2005/05/31 00:00:00
      133 ACCRUAL_PLAN_ID         8388608 661
        1 ACP_CONTINUOUS_SERVI    8388648 <NULL>-2
       65 ACP_ENROLLMENT_END_D    8388648 4712/12/31 00:00:00-1
      129 ACP_ENROLLMENT_START    8388648 2005/01/01 00:00:00-1
      193 ACP_INELIGIBILITY_PE    8388608 <NULL>-2
      257 ACP_INELIGIBILITY_PE    8388608 <NULL>-2
      321 ACP_SERVICE_START_DA         40 2003/01/01 00:00:00-1
      385 ACP_TERMINATION_DATE         40 <NULL>-2
[HSH]Indx     First     Count
--------- --------- ---------
     1767       385         1
     1947       321         1
     2004        65         1
     2187       129         1
     2342         1         1
     2479       257         1
     2607       193         1
.
.
INVAL: ACP_CONTINUOUS_SERVI    8388648         32 <NULL>
INVAL: ACP_ENROLLMENT_END_D    8388648         32 4712/12/31 00:00:00
INVAL: ACP_ENROLLMENT_START    8388648         32 2005/01/01 00:00:00
INVAL: ACP_SERVICE_START_DA         40         32 2003/01/01 00:00:00
INVAL: ACP_TERMINATION_DATE         40         32 <NULL>
.
.
[OT]Index Input/Context Name             Dtype   Value
--------- ------------------------------ ------- ------------------------------
        1 ACCRUAL_END_DATE               DATE    <NULL>
        2 EFFECTIVE_END_DATE             DATE    2005/01/09 00:00:00
        3 EFFECTIVE_START_DATE           DATE    2005/01/01 00:00:00
        4 TOTAL_ACCRUED_PTO              NUMBER  0
```

Using PYUPIP with Oracle Fast Formula

One of the best ways to debug Fast Formula is to embed trace statements directly into the formula and then to use PYUPIP to capture the trace output.

A formula function named HR_TRACE (alias HRTRACE) is already delivered that provides a mechanism for tracing out specific messages to the PYUPIP pipe.

The example formula below shows how this function can be used to output lines into the PYUPIP trace (this assumes that the PYUPIP trace has been enabled prior to the formula being called):

```
/*
   Formula Name : TEST
   Description  : Test formula
   Contexts     : DATE_EARNED, ASSIGNMENT_ID
*/

default for per_first_name is ' '
default for per_last_name is ' '

errmsg = ' '

ret = hrtrace(per_first_name)
ret = hrtrace(per_last_name)
```

Enabling PYUPIP trace from within formulae

Sometimes it may be convenient to turn the PYUPIP trace on only for the duration of a single formula or a series of formulae, as opposed to enabling it at the start of a process.

Oracle HRMS delivers the following PL/SQL functions in the PQP_UTILITIES package that allow the PYUPIP utility to be turned on and off from within a formula:

```
FUNCTION set_trace_on(
    p_trace_destination         IN       VARCHAR2
   ,p_trace_coverage            IN       VARCHAR2
   ,p_error_message             OUT NOCOPY VARCHAR2
  )
RETURN NUMBER;
```

This enables the PYUPIP trace with output going to the pipe named in `P_TRACE_DESTINATION`.

```
FUNCTION set_request_trace_on(
       p_error_message OUT NOCOPY VARCHAR2)
RETURN NUMBER;
```

This enables the PYUPIP trace with output going to the pipe with name comprised of `‘REQID’||Concurrent Process ID`.

```
FUNCTION set_trace_off(
p_error_message OUT NOCOPY VARCHAR2)
RETURN NUMBER;
```

This disables the PYUPIP trace.

In order to make use of these functions in Fast Formulae it is necessary to create some user defined Formula Functions. These should have the following definitions:

Name	SET_TRACE_ON		
Data Type	Number		
Class	External Function		
Definition	PQP_UTILITIES.SET_TRACE_ON		
Parameters			
Number	**Parameter Name**	**Type**	**Class**
1	P_TRACE_DESTINATION	Text	Input Only
2	P_TRACE_COVERAGE	Text	Input Only
3	P_ERROR_MESSAGE	Text	Output Only

Name	SET_REQUEST_TRACE_ON		
Data Type	Number		
Class	External Function		
Definition	PQP_UTILITIES.SET_REQUEST_TRACE_ON		
Parameters			
Number	**Parameter Name**	**Type**	**Class**
1	P_ERROR_MESSAGE	Text	Output Only

Name	SET_TRACE_OFF		
Data Type	Number		
Class	External Function		
Definition	PQP_UTILITIES.SET_TRACE_OFF		
Parameters			
Number	**Parameter Name**	**Type**	**Class**

1	P_ERROR_MESSAGE	Text	Output Only

The example formula below shows how these functions can be combined to generate PYUPIP trace output from a formula:

```
/*
   Formula Name : TEST
   Description  : Test formula
   Contexts     : DATE_EARNED, ASSIGNMENT_ID
*/

default for per_first_name is ' '
default for per_last_name is ' '

errmsg = ' '

retcode = set_trace_on('TRACEOUT','F',errmsg)
ret = hrtrace(per_first_name)
ret = hrtrace(per_last_name)
retcode = set_trace_off(errmsg)
```

When this formula is run the following PYUPIP trace output is produced:

```
PYUPIP apps/apps@hremeadv TRACEOUT
connected: monitoring pipe TRACEOUT
1
2
John
Smith
```

ISOLATING FORMULA FROM THE CALLING APPLICATION

Sometimes when debugging formulae it is convenient to isolate the formula and run it separately. It can also be useful to analyze the PL/SQL code that is generated when a formula is compiled.

This section discusses these topics.

Calling Formula from PL/SQL

Sometimes when debugging formulae it is convenient to isolate the formula and run it separately.

This can be achieved using the function defined in Appendix A – Formula Test Function.

This PL/SQL function accepts the FORMULA_ID and the values of any formula contexts that are needed. The example formula above would be executed in PL/SQL using the following statement:

```
Begin
   ExecuteFF(p_formula_id    => 63208
            ,p_assignment_id => 12167
            ,p_date_earned   => sysdate);
end;
```

Determining the contexts

In order to be able to use this technique it is necessary to know the correct context types that need to be passed into the formula, the following SQL provides this information:

```
select fdi.item_name
from ff_fdi_usages_f fdi
where fdi.formula_id = {formula Id}
and   fdi.usage = 'U';
```

In order to replicate the behaviour of the formula in the calling application, it will be necessary to pass in the same context values as used in the calling application.

In Appendix B there is a formula function named GetContext. This can be added to the formula in the calling application to provide PYUPIP trace output that includes the contexts and their values. Alternatively the FND_LOGGING described above can also be used to determine this information.

Debugging Formula Using the Compiled Package

There are 2 types of PL/SQL package that are generated from Fast Formula, one is generated when the formula is compiled the form or the FFXBCP command, the other is generated when the Generate Formula Wrapper Formula concurrent process is used.

Formula Execution Package

When a formula is compiled either through the form or the FFXBCP command a PL/SQL package is generated that is used when a formula is executed from PL/SQL. Its name will be made up as follows:

FFP + Formula ID + _ + DDMMYYYY e.g. FFP63208_17052005

If however a compilation error occurs in the Formula Definition form the package will not be created in the database. In order to generate the package for further analysis the following command needs to be used (N.B. the –k switch keeps the PL/SQL package even if errors are detected):

```
FFXBCP username/pwd@db 0 Y -k %% FormulaName
```

An example formula and its corresponding PL/SQL package is show in Appendix C – Example PL/SQL package including pointers to the various sections of code and how they relate the formula being generated.

If a formula compilation error occurs which cannot be resolved in the Formula Definition form then analyzing the PL/SQL formula execution package can help to diagnose the problem. Useful techniques include the following :

- Adding further trace statements
- Extracting individual SQL statements that have been generated to derive the values of any database items
- Isolating function calls for closer scrutiny.

PL/SQL Formula Wrapper Package

Oracle supplies a package header named ff_wrapper_main_pkg (was ff_wrapper_pkg), known as the 'Wrapper Package'. The Generate Formula Wrapper concurrent program creates the body for this package. This package is the 1st package that is called by the PL/SQL Execution Engine to execute a formula unless the FF_USE_PLSQL_WRAPPER pay action parameter is set to N.

At the same time the Generate Formula Wrapper concurrent program also generates a set of packages that correspond to the FFP packages described above. These are generated with the name format as follows:

FFW + Formula ID + _ + DDMMYYYY e.g. FFW63208_17052005

The Wrapper Package calls the FFW package and the FFW package itself calls the corresponding FFP package. The FFW package is created as an intermediary package to ensure that the Wrapper Package is not invalidated if a formula becomes uncompiled (which results in the FFP package being removed).

The Wrapper package is mentioned here for completeness but is not particularly useful when debugging formula issues.

Line numbers in formulae

When the formula engine raises an error it generally indicates which line the error occurred on. If the formula is not particularly big it will be easy to find the line, however if the formula is long it can be arduous to find the specific line.

As an alternative it is sometimes possible to correlate the code generated in the PL/SQL execution package to the lines in the Fast Formula. The PL/SQL package contains a local variable (`L_ PLS_INTEGER`) that is set to the corresponding Fast Formula line number ahead of the PL/SQL processing that implements the formula functionality.

FORMULA ERRORS

Common Formula Compilation Errors

Compile-time errors occur when attempting to save the formula from the Write Formulas window or the Bulk Compile Formulas process. An issue with an underlying formula function or database item typically causes these errors.

The following list of errors shows the most common types of issue:

Error: Unknown Function used in statement

```
APP-FF-33008: Unknown function 'FN' used in statement at line 7.

This may be because there is no function with the name or alias 'FN'.
Alternatively, all 'FN' functions use formula contexts that are not
available to this formula type.
```

This error occurs when a function is referenced in a formula and the formula type does not support the function contexts used.

Resolution: Determine the contexts that are supported by the formula type and ensure that the functions being called do not use contexts that are not supported.

Error: Internal Error

```
APP-FF-33980: Internal error.
A compilation error has been detected while attempting to generate
package FFP63208_16052005
Please consult your system administrator.
```

This error occurs if there is any invalid PLSQL - this can occur in the following situations:
a) function definition text is invalid
b) database item definition text is invalid
c) database item route text is invalid
d) The function used in ff_function definition is being passed the wrong number/types of parameters. This can occur if the seed data for the function is inconsistent with the plsql used in the definition text as you stated.

The term 'invalid' refers to non-existent tables, PL/SQL functions, or is syntactically invalid PL/SQL, or semantically invalid for the context in which it is being used.

Resolution: Analyse the PL/SQL function or database item being called to determine the invalid PL/SQL.

Error: The local variable was used before being initialized

```
APP-FF-33005: The local variable VAR was used before being initialized

Cause:        The variable named in the error message is being used
before any value has been assigned to it, so it has no meaningful
value.

Action:        Please ensure variables have been assigned to before
using them.
```

This error occurs when a formula references a variable that has not been initialized earlier in the formula. Although this would normally only occur when the formula is being written there are occasions when a formula references a database item and for some reason the database item cannot be accessed. This could happen if the delivery of a seeded database item failed or if the database item requires contexts that are not available for the formula type being used.

Resolution: If the variable that has not been initialized is not a database item then ensure that it is initialized in the formula. If it is a database item ensure that it exists in the database and should be available to the formula based on whether its legislation code and/or business group. Ensure that the formula type supports the database items contexts.

Common Formula Run Time Errors

Run time errors occur when a formula is actually executed. The usual cause is a data problem in the formula or the database, in addition coding problems in the formula or a function can also cause run-time errors or database item called by the formula.

The following list of errors shows the most common types of issue:

Uninitialized variables

An uninitialized local variable is one that has no value when the formula runs. This causes an error in all statements except the Return statement. For example:

```
salary = 20000
tax_band = 2000

if (tax_band < 2000) then
    tax = salary / 8
if (tax_band > 2000) then
    tax = salary / 10

if tax > 1000 then
   tax_category = '1'

return tax
```

This formula fails with an 'Uninitialized variable' message (for the variable tax) if the tax band is set to 2000.

```
ORA-20001: Variable TAX not initialized at line 10 of JRTEST2

Cause:     An attempt has been made to use a local variable before it
was initialized.  This is normally detected by the verifier, but in
certain cases this is not possible, and will result in this error.
This problem can be corrected by the altering the formula.  Assign an
initial value to the variable before it is referenced in the formula.

Action:    Please refer to your system administrator.
```

Resolution: Ensure that all local variables are initialized before being referenced in a formula.

Divide by Zero

Dividing a number by zero is an operation that provides no logical result. If this situation ever arises, Oracle Fast Formula passes a code indicating an error back to the application (the application then takes the appropriate action).

Always check for the possibility of a divide by zero error if there is any chance it could occur. For example, the formula:

```
salary = 20000
contribution_percent = 0

x = salary / contribution_percent
```

produces an error if the contribution proportion is set to zero.

```
ORA-20001: Divide by zero at line 1 of JRTEST2
Cause:     An attempt was made to divide a number by zero.  Suppose
that a formula has an input called NUMBER_IN, and a line of the form
NUMBER_OUT = 100 / NUMBER_IN.  If NUMBER_IN was passed a value of zero,
then this error would result.
Action:    Please refer to your system administrator.
```

Resolution: In this formula, check for the divide by zero condition as follows:

```
salary = 20000
contribution_percent = 0

IF   contribution_percent = 0 THEN
    (message = 'The contribution proportion is not valid.'
     RETURN message)
ELSE x = salary / contribution_percent
```

No data found

There are two typical causes for this error:

- A database item was referenced which did not return a value. This represents an error in the application data.
- A statement in a function raised a No Data Found exception and was not handled by the function. Commonly, this occurs in the GET_TABLE_VALUE function.

The first step is to determine which function or database item caused the error using the line number listed on the error message line.

Resoultion: See the sections below for discussions of troubleshooting database items and formula functions.

Too Many Rows

This error occurs when a database item returns more than one row. The steps to troubleshooting this error are the same as the No data found error. See troubleshooting database items below.

Value exceeded allowable range

This error can be caused by a statement in the fast formula or by a function call.

```
ORA-20001: Value exceeded allowable range (line 5 of JRTEST2)
Cause:     Caused by Oracle error 6502 occurring during the execution
of the formula which is raised when an arithmetic conversion error, or
```

```
string truncation error occurs.  This can also be caused by invalid
dates, for example 39-DEC-1991, 24-MAR-5000
Action:    Check your formula for possible invalid dates, numbers or
strings greater than 255 characters in length.
```

Resolution: If this error is caused by a fast formula statement, is typically due to one of the following:

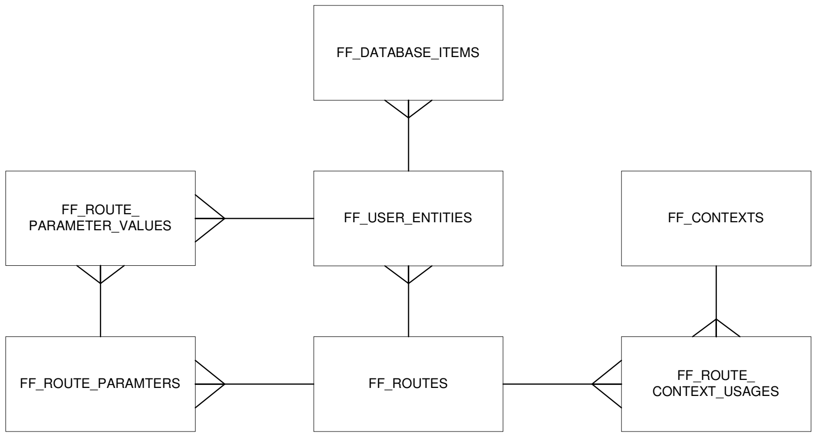
- Exceeding the maximum allowable length of a string (which is 255 characters).
- Rounding up a number to an excessive number of places, for example, round (1,100).
- Using an invalid date, for example, 39-DEC-1990.

If this is error is caused by a call to a function, it is typically due to one of the following:

- There is a mismatch between the contexts listed in the function registration and the actual PL/SQL parameters and types. As a result, the Fast Formula variables are being passed to the wrong PL/SQL parameters or types.
- The function is returning a null value. Fast Formulas expect a non-null value to be returned. Modifying the PL/SQL function to return a hard coded non-null value, then checking if the error still occurs can easily test this.

Troubleshooting Database Items

A database item is essentially a piece of SQL that is broken into a number of components, each of which is stored in the tables shown in the diagram below:



The entire definition of a Database Item can be retrieved using the script listed in Appendix D – Database Item script.

The output from the script has sections showing the SQL statement, contexts whose values are substituted into the SQL statement and parameter values that are also substituted.

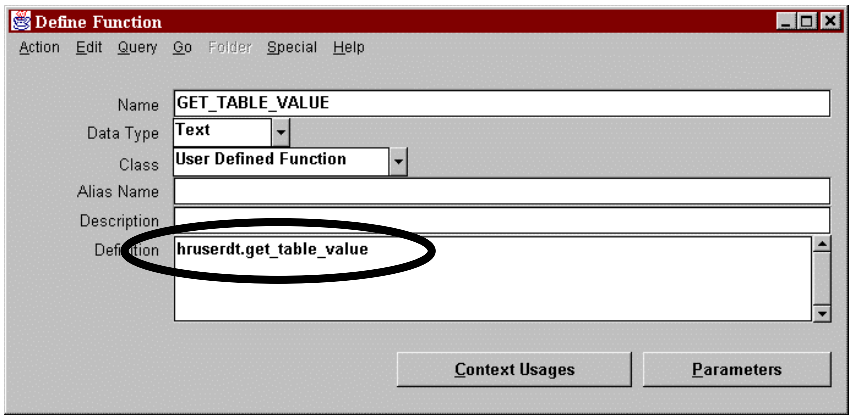
The SQL statement will have variables named &B1, &B2 etc.. When the database item is evaluated these are substituted by the formula context values. It may also have variables named &U1, &U2 etc.. These will be substituted with the parameter values shown in the output from the Database Item script.

By substituting these variables into the Database Item SQL it is possible to recreate the statement that is being executed by the formula and from there to diagnose any errors arising from the Database Item.

Troubleshooting Formula Functions

If an error is caused by a formula function, the following steps can help to determine why the function raised the error:

- Look up the function using the Define Function window (Oracle HRMS Responsibility → Other Definitions). This identifies the function definition (i.e., the PL/SQL stored function that is executed), the parameters, and the contexts:



Name	GET_TABLE_VALUE
Data Type	Text
Class	User Defined Function
Alias Name	
Description	
Definition	hruserdt.get_table_value

- Add messages to the fast formula to display the parameters and determine the context values as described above in section Determining the Contexts.

- Create a script to call the function from SQL*Plus with the identified parameters and contexts. If the parameters and contexts are correct, this will raise the same error as the formula:

```
SQL> declare
  2      result varchar2(30);
  3  begin
  4      result := hruserdt.get_table_value(101,
  5          'EMPLOYER 401K MATCH ER PAYROLL TABLE',
  6          'Percent Match',
  7          '1');
  8      dbms_output.put_line (result);
  9  end;
```

```
SQL> set serveroutput on

SQL> /
declare
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at "APPS.HRUSERDT", line 81
ORA-06512: at line 4
```

- Spool the code for the function using ALL_SOURCE and troubleshoot each SQL statement until the statement that caused the exception is identified. Replace parameters with the actual values (in this case, p_table_name and p_business_group_id were replaced):

```
SQL> select range_or_match
2  from pay_user_tables
3  where upper(user_table_name) =
4  upper('EMPLOYER 401K MATCH ER PAYROLL TABLE')
  5  and    nvl (business_group_id, 101)   = 101
  6  /

no rows selected
```

CONCLUSION

This document has covered some of the main techniques for diagnosing and debugging problems in Fast Formula. It also covered some of the common error cases and suggested ways to troubleshoot them.

If there is any additional information that you think it would be useful to include please contact HRMS Customer Services (hrms_cust_serv_uk@oracle.com).

APPENDIX A – FORMULA TEST FUNCTION : EXECUTEFF

The following PL/SQL function can be used to run a formula in isolation from its normal calling context.

```
create or replace procedure executeFF
(P_FORMULA_ID IN NUMBER
,P_BUSINESS_GROUP_ID     IN NUMBER   DEFAULT NULL
,P_PAYROLL_ID            IN NUMBER   DEFAULT NULL
,P_PAYROLL_ACTION_ID     IN NUMBER   DEFAULT NULL
,P_ASSIGNMENT_ID         IN NUMBER   DEFAULT NULL
,P_ASSIGNMENT_ACTION_ID  IN NUMBER   DEFAULT NULL
,P_DATE_EARNED           IN DATE     DEFAULT NULL
,P_ORG_PAY_METHOD_ID     IN NUMBER   DEFAULT NULL
,P_PER_PAY_METHOD_ID     IN NUMBER   DEFAULT NULL
,P_ORGANIZATION_ID       IN NUMBER   DEFAULT NULL
,P_TAX_UNIT_ID           IN NUMBER   DEFAULT NULL
,P_JURISDICTION_CODE     IN VARCHAR2 DEFAULT NULL
,P_BALANCE_DATE          IN DATE     DEFAULT NULL
,P_ELEMENT_ENTRY_ID      IN NUMBER   DEFAULT NULL
,P_ELEMENT_TYPE_ID       IN NUMBER   DEFAULT NULL
,P_ORIGINAL_ENTRY_ID     IN NUMBER   DEFAULT NULL
,P_TAX_GROUP             IN VARCHAR2 DEFAULT NULL
,P_PGM_ID                IN NUMBER   DEFAULT NULL
,P_PL_ID                 IN NUMBER   DEFAULT NULL
,P_PL_TYP_ID             IN NUMBER   DEFAULT NULL
,P_OPT_ID                IN NUMBER   DEFAULT NULL
,P_LER_ID                IN NUMBER   DEFAULT NULL
,P_COMM_TYP_ID           IN NUMBER   DEFAULT NULL
,P_ACT_TYP_ID            IN NUMBER   DEFAULT NULL
,P_ACCRUAL_PLAN_ID       IN NUMBER   DEFAULT NULL
,P_PERSON_ID             IN NUMBER   DEFAULT NULL
,P_SOURCE_ID             IN NUMBER   DEFAULT NULL
,P_SOURCE_TEXT           IN VARCHAR2 DEFAULT NULL
,P_SOURCE_TEXT2          IN VARCHAR2 DEFAULT NULL
,P_SOURCE_NUMBER         IN NUMBER   DEFAULT NULL
,P_LOCAL_UNIT_ID         IN NUMBER   DEFAULT NULL) IS
--
--  Define Local Variables
--
l_formula_id            ff_formulas_f.formula_id%TYPE;
l_effective_start_date  ff_formulas_f.effective_start_date%TYPE;
l_inputs                ff_exec.inputs_t;
l_outputs               ff_exec.outputs_t;
--
Begin
   --
   insert into fnd_sessions
   values(userenv('SESSIONID'),sysdate);
   --
   -- Initialize the formula
   --
   select formula_id,effective_start_date
   into l_formula_id,l_effective_start_date
   from ff_formulas_f
   where formula_id = p_formula_id;
   --
   ff_exec.init_formula (l_formula_id,
                         l_effective_start_date,
                         l_inputs,
                         l_outputs
                        );
   --
   if (l_inputs.first is not null) and (l_inputs.last is not null)
   then
      -- Set up context values for the formula
      for l_in_cnt in l_inputs.first..l_inputs.last
      loop
       if l_inputs(l_in_cnt).name='BUSINESS_GROUP_ID' then
            l_inputs(l_in_cnt).value := P_BUSINESS_GROUP_ID;
         end if;
       if l_inputs(l_in_cnt).name='PAYROLL_ID' then
            l_inputs(l_in_cnt).value := P_PAYROLL_ID;
         end if;
       if l_inputs(l_in_cnt).name='PAYROLL_ACTION_ID' then
            l_inputs(l_in_cnt).value := P_PAYROLL_ACTION_ID;
         end if;
       if l_inputs(l_in_cnt).name='ASSIGNMENT_ID' then
            l_inputs(l_in_cnt).value := P_ASSIGNMENT_ID;
         end if;
       if l_inputs(l_in_cnt).name='ASSIGNMENT_ACTION_ID' then
            l_inputs(l_in_cnt).value := P_ASSIGNMENT_ACTION_ID;
         end if;
       if l_inputs(l_in_cnt).name='DATE_EARNED' then
         l_inputs(l_in_cnt).value := fnd_date.date_to_canonical(P_DATE_EARNED);
         end if;
       if l_inputs(l_in_cnt).name='ORG_PAY_METHOD_ID' then
            l_inputs(l_in_cnt).value := P_ORG_PAY_METHOD_ID;
         end if;
       if l_inputs(l_in_cnt).name='PER_PAY_METHOD_ID' then
            l_inputs(l_in_cnt).value := P_PER_PAY_METHOD_ID;
         end if;
       if l_inputs(l_in_cnt).name='ORGANIZATION_ID' then
            l_inputs(l_in_cnt).value := P_ORGANIZATION_ID;
         end if;
       if l_inputs(l_in_cnt).name='TAX_UNIT_ID' then
            l_inputs(l_in_cnt).value := P_TAX_UNIT_ID;
         end if;
       if l_inputs(l_in_cnt).name='JURISDICTION_CODE' then
            l_inputs(l_in_cnt).value := P_JURISDICTION_CODE;
        end if;
       if l_inputs(l_in_cnt).name='BALANCE_DATE' then
      l_inputs(l_in_cnt).value := fnd_date.date_to_canonical(P_BALANCE_DATE);
         end if;
       if l_inputs(l_in_cnt).name='ELEMENT_ENTRY_ID' then
            l_inputs(l_in_cnt).value := P_ELEMENT_ENTRY_ID;
         end if;
       if l_inputs(l_in_cnt).name='ELEMENT_TYPE_ID' then
            l_inputs(l_in_cnt).value := P_ELEMENT_TYPE_ID;
         end if;
       if l_inputs(l_in_cnt).name='ORIGINAL_ENTRY_ID' then
            l_inputs(l_in_cnt).value := P_ORIGINAL_ENTRY_ID;
         end if;
       if l_inputs(l_in_cnt).name='TAX_GROUP' then
            l_inputs(l_in_cnt).value := P_TAX_GROUP;
         end if;
       if l_inputs(l_in_cnt).name='PGM_ID' then
            l_inputs(l_in_cnt).value := P_PGM_ID;
         end if;
       if l_inputs(l_in_cnt).name='PL_ID' then
            l_inputs(l_in_cnt).value := P_PL_ID;
         end if;
       if l_inputs(l_in_cnt).name='PL_TYP_ID' then
            l_inputs(l_in_cnt).value := P_PL_TYP_ID;
         end if;
       if l_inputs(l_in_cnt).name='OPT_ID' then
            l_inputs(l_in_cnt).value := P_OPT_ID;
         end if;
       if l_inputs(l_in_cnt).name='LER_ID' then
            l_inputs(l_in_cnt).value := P_LER_ID;
         end if;
       if l_inputs(l_in_cnt).name='COMM_TYP_ID' then
            l_inputs(l_in_cnt).value := P_COMM_TYP_ID;
         end if;
       if l_inputs(l_in_cnt).name='ACT_TYP_ID' then
            l_inputs(l_in_cnt).value := P_ACT_TYP_ID;
         end if;
       if l_inputs(l_in_cnt).name='ACCRUAL_PLAN_ID' then
            l_inputs(l_in_cnt).value := P_ACCRUAL_PLAN_ID;
         end if;
       if l_inputs(l_in_cnt).name='PERSON_ID' then
            l_inputs(l_in_cnt).value := P_PERSON_ID;
         end if;
       if l_inputs(l_in_cnt).name='SOURCE_ID' then
            l_inputs(l_in_cnt).value := P_SOURCE_ID;
         end if;
       if l_inputs(l_in_cnt).name='SOURCE_TEXT' then
            l_inputs(l_in_cnt).value := P_SOURCE_TEXT;
         end if;
       if l_inputs(l_in_cnt).name='SOURCE_TEXT2' then
            l_inputs(l_in_cnt).value := P_SOURCE_TEXT2;
         end if;
       if l_inputs(l_in_cnt).name='SOURCE_NUMBER' then
            l_inputs(l_in_cnt).value := P_SOURCE_NUMBER;
         end if;
       if l_inputs(l_in_cnt).name='LOCAL_UNIT_ID' then
            l_inputs(l_in_cnt).value := P_LOCAL_UNIT_ID;
         end if;
      end loop;
   end if;
   --
   -- Run the formula
   --
   ff_exec.run_formula (l_inputs ,
                        l_outputs
                       );
   --
   if l_outputs.count > 0 then
      for l_in_cnt in l_outputs.first..l_outputs.last
      loop
         hr_utility.trace(l_outputs(l_in_cnt).name || ' = '||
                          l_outputs(l_in_cnt).value);
      end loop;
   end if;
   --
End executeFF;
--
/
commit;
exit;
```

APPENDIX B – GETCONTEXT FUNCTION

Each formula type has a different set of contexts that it supports. The set of contexts available for a particular formula type can be determined using the following SQL:

```
select c.context_name,ft.formula_type_name
from ff_contexts c
,    ff_ftype_context_usages fcu
,    ff_formula_types ft
where ft.formula_type_name = 'QuickPaint'
and   ft.formula_type_id = fcu.formula_type_id
and   fcu.context_id = c.context_id;
```

A function that lists out the contexts passed into it must take into account the available contexts, so for instance a GetQuickpaintContexts function would only support the BUSINESS_GROUP_ID, DATE_EARNED and ASSIGNMENT_ID contexts.

The example function below works for QuickPaint formula types but would need to be modified to take into account the contexts supported by formula types other than QuickPaint.

```
CREATE OR REPLACE FUNCTION GetQuickpaintContexts
(
BUSINESS_GROUP_ID    IN NUMBER   DEFAULT NULL,
ASSIGNMENT_ID        IN NUMBER   DEFAULT NULL,
DATE_EARNED          IN DATE DEFAULT NULL,
)
--
RETURN varchar2 IS
--
PROCEDURE message(text varchar2
                 ,value varchar2) is
BEGIN
   IF value is not null then
      hr_utility.trace(text ||'-'||value);
   END IF;
END;
--
PROCEDURE message(text varchar2
                 ,value number) is
BEGIN
   Message(text,fnd_number.number_to_canonical(value));
END;
--
PROCEDURE message(text varchar2
                 ,value date) is
BEGIN
   Message(text,fnd_date.date_to_canonical(value));
END;
--
BEGIN
   message('BUSINESS_GROUP_ID',BUSINESS_GROUP_ID);
   message('ASSIGNMENT_ID',ASSIGNMENT_ID);
   message('DATE_EARNED',DATE_EARNED);

   return '0';

EXCEPTION
          WHEN OTHERS THEN
                    RETURN 'ERROR';
END GetQuickpaintContexts;
```

The following formula function needs to be created to correspond to the above PL/SQL function:

Name	GetQuickPaintContexts
Data Type	Text

Class	External Function		
Definition	GetQuickpaintContexts		
Parameters			
Number	Parameter Name	Type	Class
1	BUSINESS_GROUP_ID	Number	Input Only
2	ASSIGNMENT_ID	Number	Input Only
3	DATE_EARNED	Date	Output Only

The function is then used as follows in the formula:

```
returnCode = GetQuickPaintContexts()
```

APPENDIX C – EXAMPLE PL/SQL PACKAGE

The following form generates the PL/SQL package shown below:

```
inputs are test_input (text)

default for per_first_name is ' '
default for per_last_name is ' '

full_name = per_first_name + ' ' + per_last_name

return full_name
```

```
PACKAGE BODY FFP63208_16052005 AS
/*
Code generated by Oracle FastFormula - do not edit. Formula Name:
JRTEST2
*/
PROCEDURE DD(V IN OUT NOCOPY DATE,I IN NUMBER,D IN DATE) IS
BEGIN IF I=-2 OR I=0 THEN V:=D; END IF; END DD;
PROCEDURE DN(V IN OUT NOCOPY NUMBER,I IN NUMBER,D IN NUMBER) IS
BEGIN IF I=-2 OR I=0 THEN V:=D; END IF; END DN;
PROCEDURE DT(V IN OUT NOCOPY VARCHAR2,I IN NUMBER,D IN VARCHAR2) IS
BEGIN IF I=-2 OR I=0 THEN V:=D; END IF; END DT;
PROCEDURE FORMULA IS

L_D FF_WRAPPER_PKG.T_DATE:=FF_WRAPPER_PKG.G_D;
L_T FF_WRAPPER_PKG.T_TEXT:=FF_WRAPPER_PKG.G_T;
L_N FF_WRAPPER_PKG.T_NUMBER:=FF_WRAPPER_PKG.G_N;
L_I FF_WRAPPER_PKG.T_NUMBER:=FF_WRAPPER_PKG.G_I;
L_ERCD NUMBER;
L_ERLN NUMBER;
L_ERMT VARCHAR2(255);
BEGIN
FORMULA(L_D,L_N,L_T,L_I,L_ERLN,L_ERCD,L_ERMT);
FF_WRAPPER_PKG.G_D:=L_D;
FF_WRAPPER_PKG.G_I:=L_I;
FF_WRAPPER_PKG.G_N:=L_N;
FF_WRAPPER_PKG.G_T:=L_T;
FF_WRAPPER_PKG.G_FFERLN:=L_ERLN;
FF_WRAPPER_PKG.G_FFERCD:=L_ERCD;
FF_WRAPPER_PKG.G_FFERMT:=L_ERMT;
END FORMULA;
PROCEDURE FORMULA(D IN OUT NOCOPY FF_WRAPPER_PKG.T_DATE,
N IN OUT NOCOPY FF_WRAPPER_PKG.T_NUMBER,
T IN OUT NOCOPY FF_WRAPPER_PKG.T_TEXT,
I IN OUT NOCOPY FF_WRAPPER_PKG.T_NUMBER,
FFERLN OUT NOCOPY NUMBER,FFERCD IN OUT NOCOPY NUMBER,FFERMT OUT NOCOPY VARCHAR2)
 IS

BEGIN
FORMULA
(
T(2),
I(2),
D(1),
T(3),
I(3),
T(1),
I(1),
N(1),
FFERLN,
FFERCD,
FFERMT
);
END FORMULA;
PROCEDURE FORMULA (
T2 IN OUT NOCOPY VARCHAR2,
I2 IN OUT NOCOPY NUMBER,
D1 IN OUT NOCOPY DATE,
T3 IN OUT NOCOPY VARCHAR2,
I3 IN OUT NOCOPY NUMBER,
T1 IN OUT NOCOPY VARCHAR2,
I1 IN OUT NOCOPY NUMBER,
N1 IN OUT NOCOPY NUMBER,
FFERLN OUT NOCOPY NUMBER,
FFERCD IN OUT NOCOPY NUMBER,
FFERMT OUT NOCOPY VARCHAR2) IS
/* JRTEST2*/
LEMT VARCHAR2(255);
L_ERCD NUMBER(15,0);
L_ PLS_INTEGER;
R_ VARCHAR2(255);
BEGIN
DECLARE
EX1  EXCEPTION;
NULL_FOUND  EXCEPTION;
tv0 VARCHAR2(255);
tv1 VARCHAR2(255);
CURSOR C0 IS
SELECT
```

Overloaded procedures used in the FF execution engine

The procedure that actually performs the formula logic

```
PEOPLE.first_name,
PEOPLE.last_name,
'FF ROUTE:SEH_PER_PERSON_DETAILS_PERF'
FROM
 per_all_people_f PEOPLE
 , per_person_types PTYPE
 , per_phones PHONE
 , fnd_sessions SES
 , hr_lookups a
 , hr_lookups b
 , hr_lookups c
 , hr_lookups d
 , per_all_assignments_f ASSIGN
 where D1 BETWEEN ASSIGN.effective_start_date
 AND ASSIGN.effective_end_date
 and ASSIGN.assignment_id = N1
 and PEOPLE.person_id = ASSIGN.person_id
 and D1 BETWEEN PEOPLE.effective_start_date
 AND PEOPLE.effective_end_date
 and PTYPE.person_type_id = PEOPLE.person_type_id
 and PHONE.parent_id (+) = PEOPLE.person_id
 AND PHONE.parent_table (+)= 'PER_PEOPLE_F'
 and PHONE.phone_type (+)= 'W1'
 AND D1 BETWEEN NVL(PHONE.date_from(+),D1)
 AND NVL(PHONE.date_to(+),D1)
 and a.lookup_type = 'YES_NO'
 and a.lookup_code = nvl(PEOPLE.current_applicant_flag,'N')
 and a.application_id = 800
 and b.lookup_type = 'YES_NO'
 and b.lookup_code = nvl(PEOPLE.current_npw_flag,'N')
 and b.application_id = 800
 and c.lookup_type = 'YES_NO'
 and c.lookup_code = nvl(PEOPLE.current_employee_flag,'N')
 and c.application_id = 800
 and d.lookup_type = 'REGISTERED_DISABLED'
 and d.lookup_code = nvl(PEOPLE.registered_disabled_flag,'N')
 and d.application_id = 800
 and SES.session_id = USERENV('SESSIONID');
BEGIN
L_ERCD:=0;
LEMT:='OB';
L_:=6;
DT(T1,I1,' ');
IF I1=0 OR I2=0 THEN
BEGIN
LEMT:='PER_FIRST_NAME';
OPEN C0;
FETCH C0 INTO tv0,tv1,R_;
IF C0%notfound THEN CLOSE C0; RAISE NO_DATA_FOUND; END IF;
CLOSE C0;
IF tv0 IS NULL THEN
I1:=-2;
ELSE I1:=-1; T1:=tv0; END IF;
IF tv1 IS NULL THEN
LEMT:='PER_LAST_NAME'; RAISE NULL_FOUND;
ELSE I2:=-1; T2:=tv1; END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN L_ERCD := 3; RAISE NO_DATA_FOUND;
END;
END IF;
T3:=(T1 || ' ' || T2);
I3:=-1;

L_:=8;
I3:=1;
GOTO FFX;
<<FFX>>
NULL;
EXCEPTION
WHEN EX1 THEN L_ERCD := 1;
WHEN ZERO_DIVIDE THEN L_ERCD := 2;
WHEN NO_DATA_FOUND THEN L_ERCD := 3;
WHEN TOO_MANY_ROWS THEN L_ERCD:=4;
WHEN VALUE_ERROR THEN L_ERCD:=5;
WHEN INVALID_NUMBER THEN L_ERCD:=6;
WHEN NULL_FOUND THEN L_ERCD:=7;
WHEN HR_UTILITY.HR_ERROR THEN
BEGIN
  LEMT:=SUBSTRB(HR_UTILITY.GET_MESSAGE,1,255);
  L_ERCD:=8;
END;
WHEN OTHERS THEN
IF SQLCODE = 1 THEN L_ERCD:=-6510;
ELSE L_ERCD:=SQLCODE; END IF;
LEMT:=SUBSTRB(LEMT||' '||SQLERRM,1,255);
END;
FFERLN:=L_; FFERCD:=L_ERCD; FFERMT:=LEMT;
END FORMULA;
END
FFP63208_16052005
;
```

SQL to evaluate the Database Items

Formula logic

Error Handling

APPENDIX D – DATABASE ITEM SCRIPT

The following script can be used to list out the definition of a database item:

```
set pagesize 999
set heading off
set array 1
set long 30000
set feedback off
set verify off
set echo off
set embedded on
column dtxt format a80
column parameter_name format a15
column value format a40
spool route
pro DBI.DEFINITION_TEXT / ROUTE_TEXT____________________
select 'Select '||DEFINITION_TEXT ,
'/* dbi.data_type('||dbi.data_type||')*/' dtxt,
'/* ue.not_found_allowed('||ue.NOTFOUND_ALLOWED_FLAG||')*/' dtxt,
'/* dbi.null_allowed_flag('||Dbi.NULL_ALLOWED_FLAG||')*/'dtxt,
'/* ue.leg_code/bg('||nvl(ue.legislation_code,ue.business_group_id)||')*/' dtxt,'/*
route_name('||r.route_name|| ')*/ from ' dtxt, r.text
 from ff_database_items dbi,ff_user_entities ue,
      ff_routes r
where ue.user_entity_id = dbi.user_entity_id
and user_name like upper('&&user_name')
and ue.route_id = r.route_id(+)
/
pro
pro FF_ROUTE_CONTEXT_USAGES__________________
set heading on
select r.route_id, sequence_no, context_name from ff_route_context_usages rcu ,
ff_database_items dbi,ff_user_entities ue,
      ff_routes r, ff_contexts c
where ue.user_entity_id = dbi.user_entity_id
and user_name like upper('&&user_name')
and ue.route_id = r.route_id(+)
and r.route_id = rcu.route_id
and rcu.context_id = c.context_id
order by 1
/
pro
pro FF_ROUTE_PARAMETER_VALUES________________
select sequence_no, parameter_name, value from ff_route_parameters rp,
ff_database_items dbi,ff_user_entities ue, ff_route_parameter_values rpv,
      ff_routes r
where ue.user_entity_id = dbi.user_entity_id
and user_name like upper('&&user_name')
and ue.route_id = r.route_id(+)
and r.route_id = rp.route_id
and rpv.route_parameter_id = rp.route_parameter_id
and rpv.user_entity_id = ue.user_entity_id
order by 1
/
spool off
undefine user_name
```

APPENDIX E – RELATED DOCUMENTS

The following related documents could be found on Metalink:

Note 296833.1 - Troubleshooting Guide for Fast Formula issues in Benefits

Note 218059.1 - Oracle Fast Formula Reference Guide for Standard and Advanced Benefits

Note 223280.1 - Using Oracle Fast Formula for creating fast formulas

Note 160469.1 - R11.5 How to Setup and Run PYUPIP on a Form

Note 260552.1 - Understanding PTO changes (Accrual changes) since HRMS PF_D

Note 211422.1 - Fast Formula FAQ

Note 108341.1 - An Introduction to HR: System Profiles

Note 256842.1 - How to Display Debug Messages in Payroll Fast Formula

Debugging HRMS Fast Formula
May 2005
Author: John Rhodes
Contributing Authors:

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

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