Page 1
January 2006 Volume II Issue 1
ORAtips Journal

Batch Invoice Uploads into Oracle® Payables Using MS Excel

By Kevin Ellis

Editor's Note: MS Excel to load invoice batches to Oracle Payables! Even Oracle ADI doesn't do that, in fact few 3rd-party products do. What to do? Do as Kevin Ellis has done, develop your own in-house invoice batch interface. And if that isn't enough to pique your interest, how about an interface for keeping your chart of accounts current between production legacy and Oracle systems? He has done that too. And he has shared all in this comprehensive ORAtips cover story, discussing custom concurrent program and UNIX shell script development, Oracle Applications set-up, and end-user considerations.

Introduction

In the inaugural volume of ORAtips, I presented an article titled "Real-Time Migration of Oracle Application Setups Done In-House". In that article, I gave a little background on the origin of Oracle Applications (Finance) at Humana. Emphesys was to have been the name of a spin-off company that never happened." The rollout of Oracle Application (Finance) was to be the book of record for the new spin-off. As the spin-off never happened, the implementation of Oracle Applications was retained as a sub-ledger to the corporate legacy books. Emphesys was reorganized into product development. Later as time passed, senior leadership decided that the corporate book of record (legacy) would eventually need to be enhanced or converted. The long-term goal was conversion. Better stated, Oracle Applications (Finance) would eventually be the book of record for the corporation. However, conversion would be done in parts. The first would be the conversion of the legacy accounts payable system.

The challenge associated with any type of conversion is ensuring end-user buy-in.

In this article, I will discuss what I believe is the most visible custom interface that has been deployed as part of our Oracle Application (Finance) release. Its origin came from a major project some three years ago that converted the corporate legacy accounts payable system to the Oracle Applications (Finance) Payables module.

As with any major conversion, system functionality currently used must be retained. The challenge was deploying an Oracle Applications (Finance) batch invoice system that mirrored the legacy system. A second challenge, associated with any type of conversion, is ensuring end-user buy-in. Not only do we need their acceptance of the new interface, but also their willingness to use the interface in a completely new financial management system. It was important to be very conscious of the end users' reactions; in other words, the fear of change. The goal was to give the end users an intuitive system, easing their fears and empowering them to have control of both their job and the system.

The interface, better known as "Invoice Upload Interface", is highly visible because it is heavily used every day by multiple end users. It has been in production use for three years. In the beginning, its purpose was to allow multiple end users to process invoices in real-time (or parallel processing of invoice batches). As the end users began to recognize the power of the interface, additional enhancements were requested. Some of these enhancements were:

- One-Time Vendor Invoice Uploads
- Match to PO Invoices Uploads
- Sharing the Process with disconnected Business Units within Humana
- Automate the process to include Mass Additions of Assets

Discussing each of the enhancements listed above is a paper unto itself. What I will discuss is the initial deployment of the original "Invoice Upload Interface" and the tools required to do it.

Process Requirements

The end users wanted to replicate how invoice batches were processed in the legacy system. The legacy pro-

ORAtips © 2006 Klee Associates, Inc.

cess involved screens. At first thought, it would be reasonable to assume our solution would be deployment of a sub-system in Oracle Applications using Oracle Forms. However, there were multiple problems with this. Our development staff members are technology rookies and were in fact taking Oracle Forms training classes during this project. That coupled with a tight budget and deadline constraints also eliminated bringing in external consultants to assist.

During the analysis phase, it was noted the end users were comfortable using Microsoft Excel, as were the System Administrators, who are also accounts payable experts. My years of working with Oracle products had helped me learn the technical side of Oracle Payables, and I have a good working knowledge of Excel. This triggered the idea of creating an Oracle Payables business template in Excel (administered by our functional system administrators) as the key to this challenge. A concurrent program could be used to process and load the template into Oracle Payables via an open interface. This solution would meet two major requirements: ease of use and limited training.

Still, there were other hurdles to overcome: business requirements. One has already been discussed: multiple end users running the process in parallel. To meet this requirement, our design was simple enough, and this is discussed further in the System Design section. At a quick glance:

1. A batch number uniquely identifies each batch.
2. The end users save the Excel template batches as comma-delimited files (or CSV files).
3. The system uses a CSV file format to load data into tables via SQL*Loader.

This solution allowed the interface to be executed multiple times in parallel.

As previously mentioned, Oracle Applications (Finance) was to be rolled out as part of the spin-off. The chart of accounts (COA) structure differed in Oracle Applications (Finance) from that of the legacy system. It was concluded that the mapping between Oracle and legacy COA combinations would be maintained on the legacy side of the business. Since the end users required the ability to continue booking invoices using the legacy COA structure, a custom interface was required to pull the COA mappings down so that the invoices could be booked to the correct COA combination in Oracle. Given that the COA combinations are changed daily, the interface (COA mappings) would need to be scheduled to run daily so that the Oracle mappings remained current. In the event of an emergency mapping, the interface also needed to be flexible so that it could be executed on demand.

Excel is used as a substitute for the legacy system and is the data entry point.

Our solution was to develop a sub-system that mimics legacy system functionality. Excel is used as a substitute for the legacy system and is the data entry point. A process is deployed to maintain COA mappings between the legacy system and Oracle, scheduled to run daily with the ability to be executed on demand. A concurrent program is deployed, which can be submitted multiple times simultaneously for different batches to take care of translating the legacy COA values to Oracle COA combinations. The actual mappings are managed and maintained on the legacy system.

System Design

System design is fairly simple from the standpoint of the end user:

- A predefined Excel template for data entry
- A location on the server to store the Excel template batches
- A concurrent program that would select, process, and load the invoices stored in the Excel template batches into Oracle Payables

The concept is straightforward. The details are another story. Ownership of the design for the Excel template was given to the functional staff since they were accounts payable experts, plus had a comfort level designing an Excel template. Granted, some negotiation was necessary. For instance, the technical team knew Oracle processed invoices in a drill down sort of fashion. Batches are composed of invoices and invoices are composed of lines. Also, consideration for maintaining the COA mappings had to be factored in. Figure 1 demonstrates (from a business perspective) the design of the system.

ORAtips © 2006 Klee Associates, Inc.

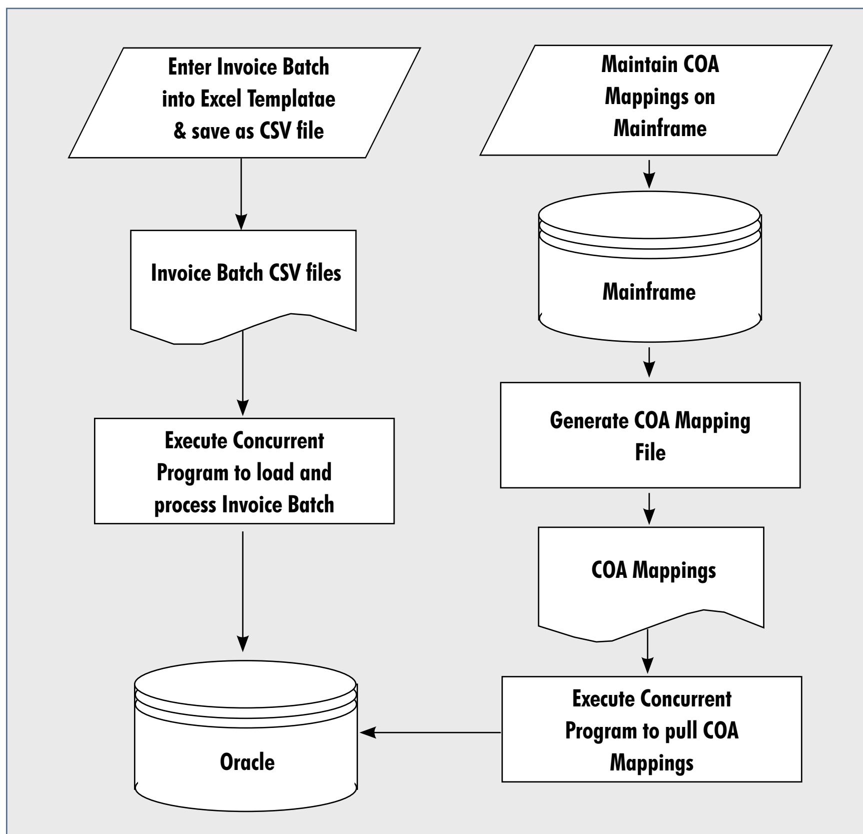

Figure 1: Business System Design

Enter Invoice Batch is done via a pre-defined Excel template. Once the template is populated, it is saved as a comma-delimited file to a specific directory on a specific network server. The file name includes the batch number as determined by the end user. The number is used as a run-time parameter when executing the concurrent program to pull the comma-delimited file from the network server to UNIX. Not only does the batch number allow the concurrent program to identify the correct data file to pull, but also allows other end users to process different batches at the same time. In other words, the system is designed to allow parallel processing of invoice batches.

Because the end users use the corporate COA structure to enter invoice batches, the COA mappings must be pulled from the mainframe to Oracle for proper accounting of the invoices in the Oracle Payables system. A process on the mainframe generates the required data, the COA mapping file. In Oracle, a concurrent program pulls the COA mappings to Oracle for reference when processing invoice batches.

Figure 2: Excel Template, Part 1

At this point, I will discuss the details of the system. Please note that I will not elaborate on the COA mapping portion since it may only be unique to our company and may not even be necessary for yours. Even if it were relevant to your company, the structures would differ. Additionally, the discussion of the COA mappings takes away from the major goal of this article, which is the processing of the invoices in batch format

The MS Excel Template

As mentioned previously, the data entry point is the Excel template. At Humana, we designed the template with two goals in mind: items that pertained to the business and those that are required to be processed via Oracle Payables.

The first three lines of the template are reserved. The first line of the template is header labels for the batch. Basic batch level information is entered into the second line. The third line contains invoice line labels, identifying the columns for each data element to be entered for an invoice line. The bulk of the data (invoice lines) are entered starting at line 4 forward. Figures 2 and 3 are examples of our Excel template.

Line 2 contains the basics about the batch. The end user enters the

ORAtips © 2006 Klee Associates, Inc.

Page 4
January 2006 Volume II Issue 1
ORAtips Journal

batch number along with the batch control total, batch date, accounting period, the batch control count, and their Oracle user id (created by). The other batch header columns are calculated fields. In other words, Excel computes the accumulated batch total and the difference displayed in the column titled "Batch Variance". Likewise, Excel calculates the "Accumulated Batch Count" field and the variance displayed in the field titled "Batch Count Variance". Additionally, there is one other calculated field called "Invoice Count", which tells the user (once all invoice lines have been entered) the number of unique invoices in the batch.

At the invoice line level, the end user enters data about each invoice line included in the batch. Data related to each specific invoice is denoted by the burgundy color of the header column. Likewise, invoice line level data is denoted by the yellow color of the header column. You may notice that there are column headers listed in blue. These items represent an enhancement that was applied to the interface related to performing automated mass additions. These attributes are optional since they are only relevant to invoices involving assets. With every invoice, the following is required:

- Batch number
- Date the invoice was received
- The invoice date
- Invoice number
- Vendor identification (either a number or code)
- Special handle
- Pay alone
- Terms

Figure 3: Excel Template - Part 2

Other information (such as foreign PO number, invoice description comments and secure information) is optional. At the line level, the following is required:

- Line number
- COA information (COID & UDN/Acct)
- Distribution amount

….. data entered for state, county or local tax will create an additional invoice line…

Likewise, there is optional information (such as state tax, county tax, local tax, distribution description, major & minor category, tag number, serial number, and expense UDN/Acct). It should be noted that data entered into state, county, or local tax will result in an extra invoice line being created. In other words, each invoice line listed in the Excel spreadsheet will create from one to four invoice line entries in Oracle Payables pending tax information rendered.

The Custom Concurrent Program

The custom concurrent program executable is built using UNIX script, which will be discussed further in the UNIX Shell Program Design section. The custom concurrent program calls the custom executable. The custom concurrent program setup is displayed in Figure 4.

The concurrent program is composed of several parameters, most of which are invisible to the end user. However, one is visible and required: batch number. All of the parameters are displayed in Figures 5 and 6. For the batch number, a custom value set was deployed (Figure 7). There is nothing special about this value set other than it defines the batch

ORAtips © 2006 Klee Associates, Inc.

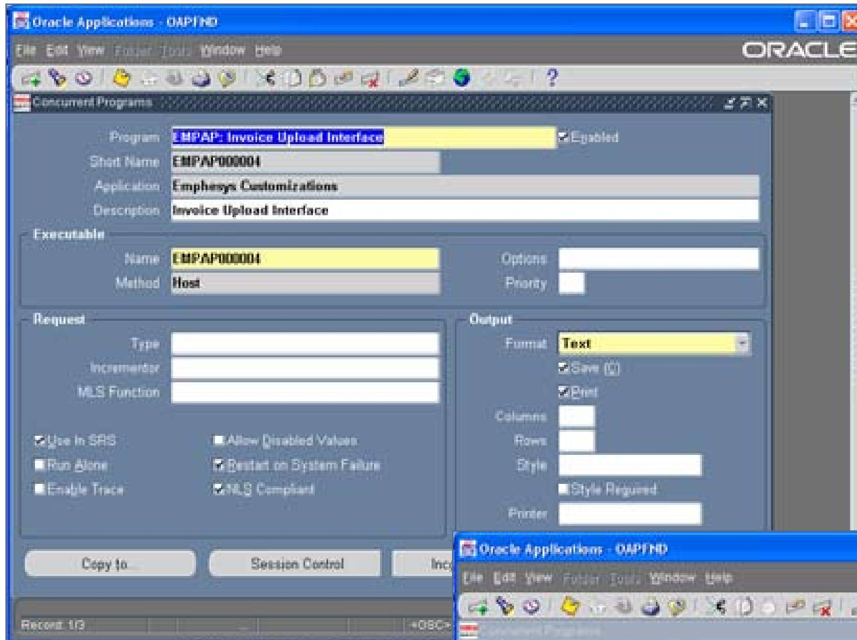

Figure 4: Custom Concurrent Program

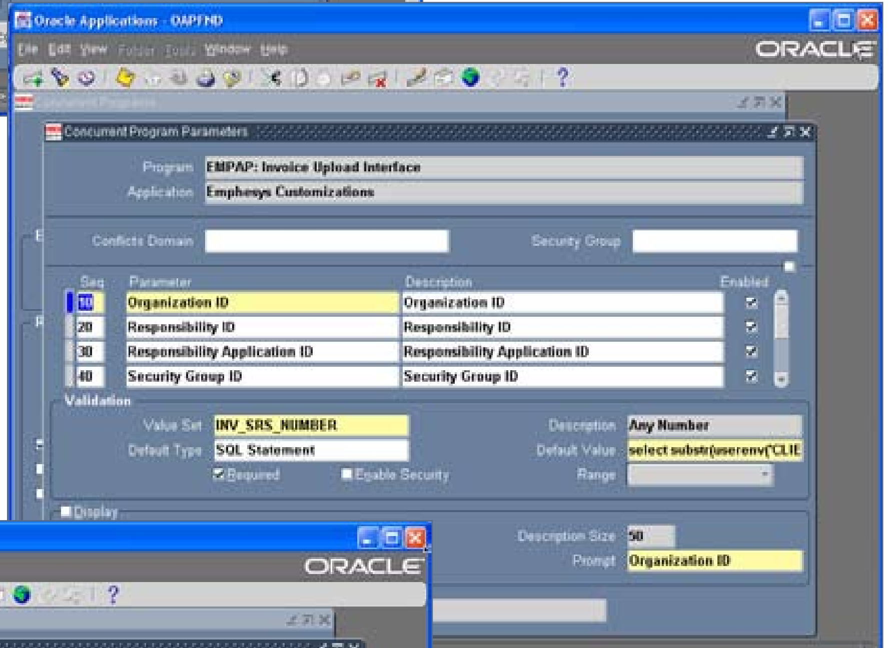

Figure 5: Parameters Part 1

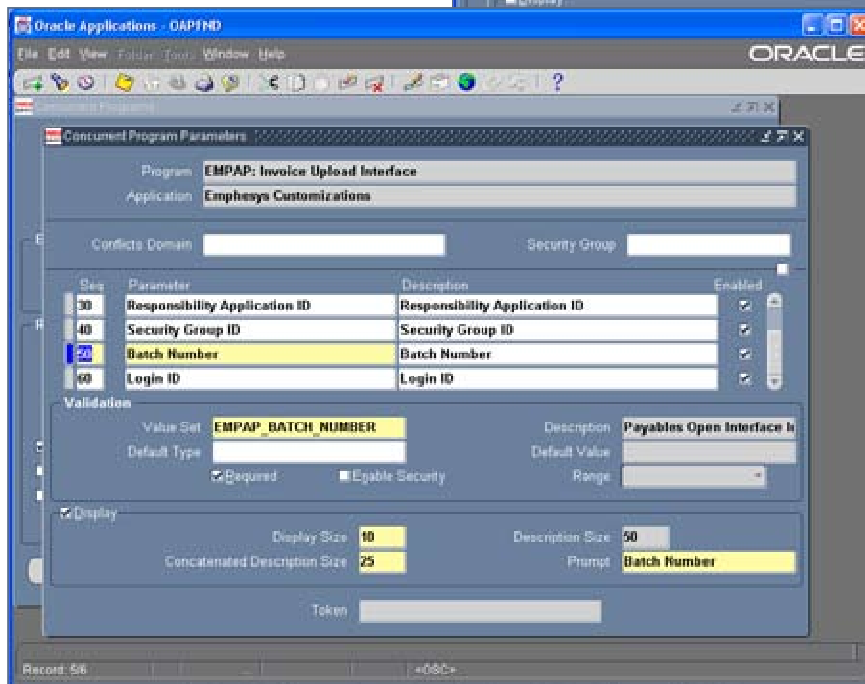

Figure 6: Parameters Part 2

ORAtips © 2006 Klee Associates, Inc.

number as an alpha-numeric string of up to ten characters in length with no validation. It is possible that this rule may change in the future. The UNIX script uses all of the other parameters.

The UNIX Shell Program Design

What we have seen up to this point is the cosmetic portion of the interface. Now comes the guts: the background process. The data flow of the background process is illustrated in Figure 8. The first part of the process is simple: pull over the invoice batch data file from the NT server to UNIX. This is accomplished via a FTP session. Because the interface must be able to process multiple batches at the same time, the file name of each batch contains the batch number.

Pulling the Data File to UNIX

In our deployment, UNIX was used to drive the entire interface. The concurrent program initiates a call to a UNIX shell program. UNIX shell programs for Oracle Applications can be set up in many ways. One way is by using a UNIX link. This is beneficial to the developer in terms of future enhancements. We did use this deployment method because at the time of development, we did not have any knowledge that UNIX shell programs could be setup in that fashion. Instead we used AWK to parse the parameter string sent from the concurrent program. The code for executing the FTP in this fashion is shown in Figure 9.

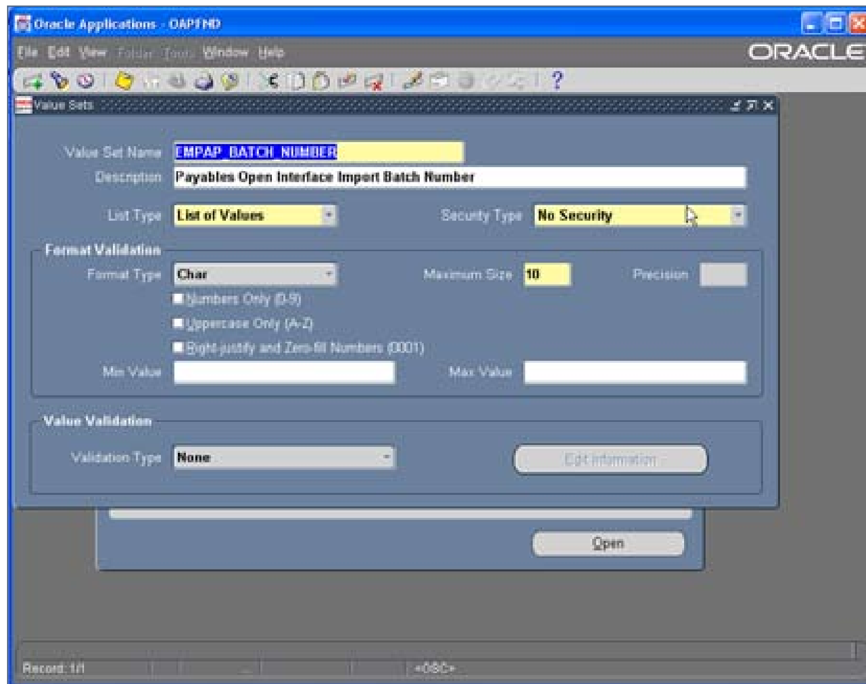

Figure 7: Value Set

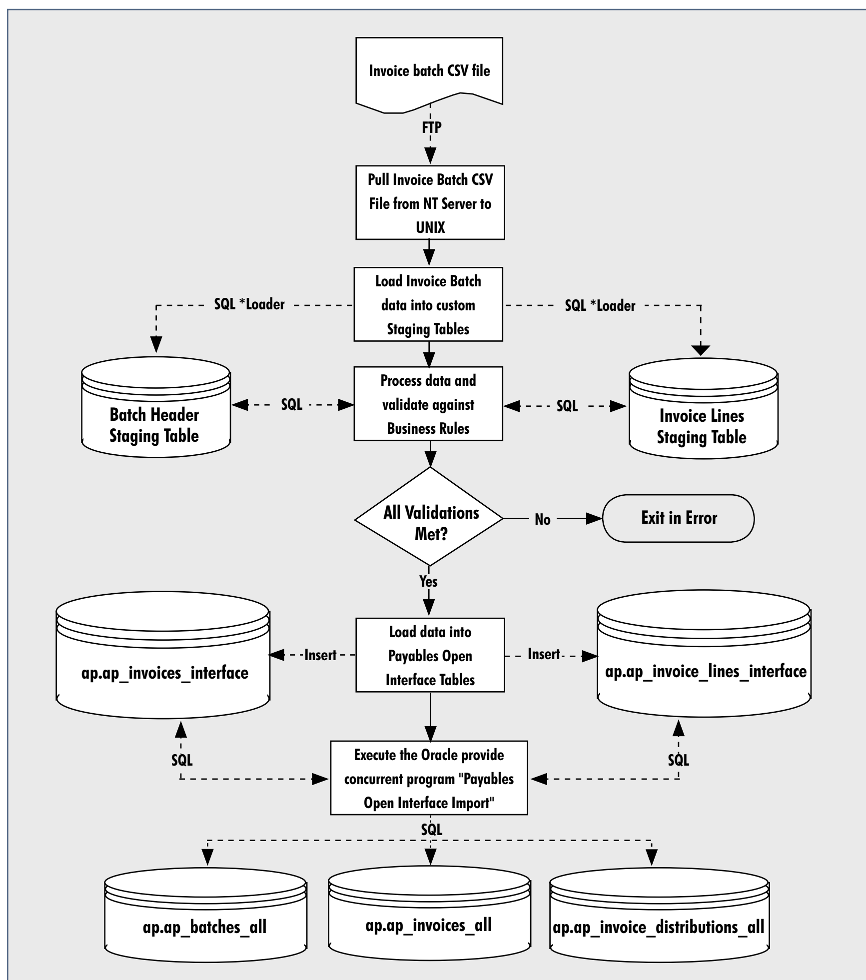

Figure 8: Background Data Flow

ORAtips © 2006 Klee Associates, Inc.

Page 7
January 2006 Volume II Issue 1
ORAtips Journal

```
#*******************************************************
#*** Retrieve Parameters from Concurrent Program ***
#*******************************************************
REQUESTID=`echo $1 | awk '{print $2}' | sed 's;FCP_REQID=;;'`
USERPASS=`echo $1 | awk '{print $3}' | sed 's;FCP_LOGIN=;;' | sed 's;";;g'`
USERID=`echo $1 | awk '{print $4}' | sed 's;FCP_USERID=;;'`
ORGID=`echo $1 | awk '{print $9}' | sed 's;";;g'`
RESPID=`echo $1 | awk '{print $10}' - | sed 's;";;g'`
RESPAPPLID=`echo $1 | awk '{print $11}' - | sed 's;";;g'`
SECGRPID=`echo $1 | awk '{print $12}' - | sed 's;";;g'`
BATCH_NUMBER=`echo $1 | awk '{print $13}' - | sed 's;";;g'`
LOGINID=`echo $1 | awk '{print $14}' - | sed 's;";;g'`

#*************************
#*** Setup Constants ***
#*************************
FILENAME_BASE="empap_humvp"
DIRECTORY_XFER="/staging"
DIRECTORY_DATA="${EMPCUST_TOP}/inbound"

#*********************************************
#*** Retrieve E*Mail Address for Caller ***
#*********************************************
EMAILADDR1=`echo "set heading off
  set echo off
  select email_address
  from apps.fnd_user
  where user_id = ${USERID};" | sqlplus -s ${USERPASS}`
EMAILADDR1=`echo ${EMAILADDR1} | sed 's;^ ;;g'`

echo "Batch Name :"${BATCH_NUMBER}

#*******************************************************
#*** Attempt to retrieve Invoice Batch Data File ***
#*** from NT Server               ***
#*******************************************************
echo "Retrieving batch data file ${FILENAME_BASE}${BATCH_NUMBER}.csv"

ftp -inv 'humana.ap.invoices.batch_upload' <<EOFTP1
user 'oracle_upload' 'password'
cd "/ap/oracle/csv"
get ${FILENAME_BASE}${BATCH_NUMBER}.csv ${DIRECTORY_XFER}/${FILENAME_BASE}${BATCH_NUMBER}.csv
del ${FILENAME_BASE}${BATCH_NUMBER}.csv
quit
EOFTP1

#*********************************************************
#*** If Data File does not exist, then send e*mail ***
#*** to caller about the problem. However, if it  ***
#*** does exist, put data file in data processing  ***
#*** directory.            ***
```

Figure 9: FTP - continued on next page

ORAtips © 2006 Klee Associates, Inc.

```
#************************************************************
ls ${DIRECTORY_XFER}/${FILENAME_BASE}${BATCH_NUMBER}.csv
if [ $? -ne 0 ];
then
        echo "Error: data file ${FILENAME_BASE}${BATCH_NUMBER}.csv does not exist. Please download invoice batch file and re-run concurrent program EMPAP: Invoice Upload Interface."
        (echo "Data file ${FILENAME_BASE}${BATCH_NUMBER}.csv does not exist. Please download invoice batch file and re-run concurrent program EMPAP: Invoice Upload Interface.")|mailx -s "EMPAP: Invoice Upload; Error, Request ID ${REQUESTID}" ${EMAILADDR1}
        exit 1
else
        echo "Data file downloaded to transfer directory."
        echo "Moving data file to data directory."
        mv ${DIRECTORY_XFER}/${FILENAME_BASE}${BATCH_NUMBER}.csv ${DIRECTORY_DATA}/${FILENAME_BASE}${BATCH_NUMBER}.csv
fi
```

Figure 9: FTP - continued from previous page

All parameters set up in the concurrent program, plus some that Oracle sends automatically (like the request ID, APPS/password connection to the Oracle Database, the user ID executing the concurrent program, the user name, selected printer, etc.), can be referenced by the first parameter ($1) in a UNIX shell program. In this example, we are only going to look at those that are pertinent to performing the FTP.

In the example, I have listed all of the parameters that have been set up by the design of the concurrent program. I wanted to point out that custom parameters are referenced in a UNIX shell program starting at the 9 position of parameter $1. In other words, to get the Org ID you must parse through to the 9 string in parameter $1 in order to obtain its value. Likewise, the 10th string in parameter $1 represents the Responsibility ID, and so forth. The first 8 strings in parameter $1 are reserved and are passed to a UNIX shell program automatically. We will use some of these to perform the logic above. Also important to note is that the batch number will be saved in a UNIX variable ${BATCH_NUMBER} and is the 13 string in parameter $1.

Once the parameters have been received from the concurrent program, constants are set up. In this case, the constants serve as placeholders for the base name for the data file that is to be pulled over, the directory that the data file will initially be placed in, and the final staging area that the data file will be placed in, provided that we were successful in pulling it over in the first place. All data files have a base name. In this case, it is "empap_humvp", standing for Emphesys Account Payable for Humana Vendor Payments. This string is stored in the constant variable ${FILENAME_BASE}. That is basically what our invoice upload process is for: processing invoices for services provided by external vendors. What follows that file name is the batch number. Since the end user saves the invoice batch as an comma-delimited file, on the remote server the file can be referenced by the concatenation of UNIX variables "${FILENAME_BASE}${BATCH_NUMBER}.csv".

Up to this point, I have demonstrated how we retrieve the concurrent program parameters and how we determine what file to pull. Now let us actually try finding it. This is performed via an FTP session. I have placed some dummy names to reference a server I call "humana.ap.invoices.batch_upload". Likewise, I assume there is an account set up for us called "oracle_upload" and that the password to this account is "password". This would not be very secure by any means, but I wanted to keep this part of the demonstration simple. I also assume that the associates save their comma-delimited invoice batch file under a directory called "/ap/oracle/csv".

Given these assumptions, I have configured the UNIX script to connect to the remote server at that point and attempt to find and pull a file that is equivalent to the value stored from referencing "${FILENAME_BASE}${BATCH_NUMBER}.csv". Assuming that the file exists, it is pulled from the remote server and placed on a directory in UNIX called "/staging". My next step is to determine if a file was received. If no file is received, I must notify the caller of the concurrent program of the problem and terminate the UNIX shell program. To terminate a UNIX shell program abnormally is accomplished using the "exit" command. This will cause the concurrent program that

ORAtips © 2006 Klee Associates, Inc.

executed it to immediately complete in error. As part of this process, I send an e-mail to the caller, stating that the concurrent program has completed in error. I also send them the request ID so that they can quickly find and review the report for the problem in question via the concurrent manager. If the file was found, then I simply move the invoice batch file to a data processing directory.

At this point you might ask me, how do you know which person to send the e-mail to. Using the 3rd and 4th strings in parameter $1, you can identify the Oracle APPS/password connection string of the User ID of the caller that executed the concurrent program. Again, I assume that the e-mail address has been set up in Oracle Applications and can be found in the AOL table called FND_USER. Starting a SQL*Plus session, I make a simple SQL select statement pulling the e-mail address for the user ID. Once the e-mail address is returned, trim it for unnecessary characters.

Load the Data into Custom Staging

Assuming that the comma-delimited file was retrieved successfully from the remote server onto UNIX, the next step can commence. The step is loading the data to custom staging tables. The design splits the data into 2 separate tables: one for header information (batch level) and the other containing the actual invoice distributions (or lines). SQL*Loader is used to accomplish this task. The control files developed for this task are shown in Figures 10 and 11.

empap_humvp_header_tl.ctl - Notepad

```
options (rows=1000000, errors=0, skip=1, load=1)
load data
        append
        into table empcustom.empap_humvp_header_tl
        fields terminated by ','optionally enclosed by '"'
        trailing nullcols
(
   batch_number                char "ltrim(rtrim(:batch_number))"
  ,batch_control_total         decimal external
  ,accumulated_batch_total     decimal external
  ,batch_total_variance        decimal external
  ,batch_date_format           char "ltrim(rtrim(:batch_date_format))"
  ,accounting_period           char "ltrim(rtrim(:accounting_period))"
  ,batch_control_count         integer external
  ,accumulated_batch_count     integer external
  ,batch_count_variance        integer external
  ,created_by                  char "ltrim(rtrim(:created_by))"
)
```

Figure 10: Header Control File

empap_humvp_lines_tl.ctl - Notepad

```
options (rows=1000000, errors=0, skip=3)
load data
  append
   into table empcustom.empap_humvp_lines_tl
   fields terminated by ','optionally enclosed by '"'
   trailing nullcols
(
   batch_number                   char "ltrim(rtrim(:batch_number))"
  ,invoice_received_date_format   char "ltrim(rtrim(:invoice_received_date_format))"
  ,invoice_date_format            char "ltrim(rtrim(:invoice_date_format))"
  ,invoice_number                 char "ltrim(rtrim(:invoice_number))"
  ,foreign_po                     char "ltrim(rtrim(:foreign_po))"
  ,line_number                    integer external
  ,vendor_number                  char "ltrim(rtrim(:vendor_number))"
  ,vendor_site                    char "ltrim(rtrim(:vendor_site))"
  ,coid                           char "ltrim(rtrim(:coid))"
  ,udn_acct                       char "ltrim(rtrim(:udn_acct))"
  ,distribution_amount            decimal external
  ,state_tax                      decimal external
  ,county_tax                     decimal external
  ,local_tax                      decimal external
  ,invoice_total                  decimal external
  ,description                    char "ltrim(rtrim(:description))"
  ,comments                       char "ltrim(rtrim(:comments))"
  ,secure_one                     char "ltrim(rtrim(:secure_one))"
  ,secure_two                     char "ltrim(rtrim(:secure_two))"
  ,special_handle                 char "upper(ltrim(rtrim(:special_handle)))"
  ,pay_alone_flag                 char "upper(ltrim(rtrim(:pay_alone_flag)))"
  ,terms_name                     char "ltrim(rtrim(:terms_name))"
  ,major_category                 char "ltrim(rtrim(:major_category))"
  ,minor_category                 char "ltrim(rtrim(:minor_category))"
  ,tag_number                     char "ltrim(rtrim(:tag_number))"
  ,serial_number                  char "ltrim(rtrim(:serial_number))"
  ,expense_udn_acct               char "ltrim(rtrim(:expense_udn_acct))"
)
```

Figure 11: Line Control File

The header control file (Figure 10) would process the header information of the invoice batch. For each batch, a single record is retrieved from the comma-delimited file. It is the second physical line in the file. Note, the first physical line contains the labels for each field associated with the batch level header information. The batch level record is loaded into a table called empap_humvp_header_tl that exists in a custom schema called empcustom. Additionally, data loaded into this table is appended. The reason is that there are custom Discoverer reports (not discussed in this article) that have been provided to the end users for reporting on current and previous invoice batch uploads.

The lines control file (Figure 11) loads all of the invoice lines for the batch in question. All invoice lines start at the physical fourth line of the comma-delimited file. The third physical line of the file contains the labels for the invoice lines that follow. The only limitations for entering invoice lines in a batch are dependent on how many rows of data Excel can handle. The invoice line records are loaded into a custom table called empap_humvp_lines_tl, which is stored in a custom schema called empcustom. Just to note, we allow for no errors to occur during load. Any errors encountered will result in the interface terminating in error.

Let's say, for instance, that the invoice batch being processed is for batch 12351. The associate respon-

ORAtips © 2006 Klee Associates, Inc.

sible for this batch would have produced a comma-delimited file called "empap_humvp12351.csv". For simplicity, let's say that the batch contains two invoices: one with three distributions and the other with two. The batch would consist of a total of five invoice lines to populate, at the minimum. It is possible (given our business design) that other invoice lines may be created. This concept will be discussed later under the topic of loading data to the Payables Open Interface tables. Figure 12 is an example of what the comma-delimited file would look like.

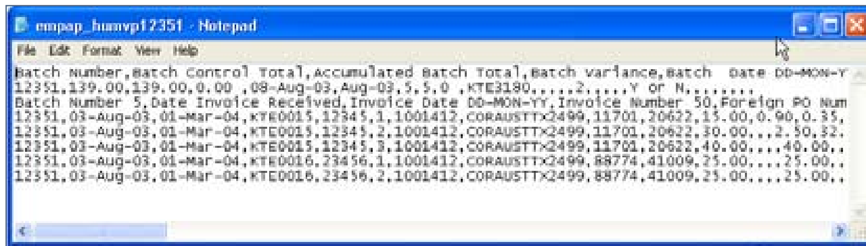

Figure 12: Comma-Delimited File

This is the same comma-delimited file that will be processed by both SQL*Loader control files. The header control file will process a single record: that which exists on the second physical line for the file. The line control file will process all invoice lines starting at the fourth physical line forward.

Let us now review how this would be executed via the UNIX shell program. Figures 13 and 14 are samples of the UNIX scripting code that exe-

```
/************************************/
/*** Set SQL*Loader Parameters ***/
/************************************/
P_CTL="${DIRECTORY_SQLLDR}/empap_humvp_header_tl.ctl"
P_DATA="${DIRECTORY_DATA}/${FILENAME_BASE}${BATCH_NUMBER}.ctl"
P_BAD="${DIRECTORY_BAD}/${REQUESTID}h.bad"
P_LOG="${DIRECTORY_BAD}/${REQUESTID}h.log"
P_DSC="${DIRECTORY_BAD}/${REQUESTID}h.dsc"

/****************************************/
/*** Display SQL*Loader Parameters ***/
/****************************************/
echo "Control file : ${P_CTL}"
echo "Log file    : ${P_LOG}"
echo "Data file   : ${P_DATA}"
echo "Bad file    : ${P_BAD}"
echo "Discard file : ${P_DSC}"

/*************************************/
/*** Execute SQL*Loader Session ***/
/*************************************/
sqlldr ${USERPASS} control=${P_CTL} data=${P_DATA} bad=${P_BAD} log=${P_LOG} discard=${P_DSC}

/*******************************************/
/*** Determine if there were problems ***/
/*******************************************/
ls ${P_BAD}
if [ $? -ne 0 ];
then
        echo "Data file loaded successfully into table empap_humvp_header_tl."
else
        echo "ERROR: Problems occurred during data load into table empap_humvp_header_tl."
        (cat ${P_BAD})|mailx -s "EMPAP: Invoice Upload Interface, Error with Concurrent Request ${REQUESTID}" ${EMAILADDR1}
        exit 2
fi
```

Figure 13: UNIX Code for Header Control File

ORAtips © 2006 Klee Associates, Inc.

cutes SQL*Loader to accomplish the data load for each custom table.

For simplicity, the first thing I do is set up SQL*Loader parameters. In the event I would have to change the settings, all I would need to do is change the values set against the parameter. The parameters are used to display results to the concurrent program executing the SQL*Loader. They can be returned to the caller (via e-mail) for review in the event an error occurs. One important note about the listing above, all of the output files that are generated after executing SQL*Loader are denoted by the concurrent request ID followed by the letter "h". This signifies that the file is related to SQL*Loader header results for the given Concurrent Request ID. If a problem occurs during the data load, a bad file is generated. If that file exists, an error message e-mail is sent to the caller and the concurrent program is terminated in error.

The lines SQL*Loader process is very similar in most respects. The major difference is the control file being executed along with the names of the potential files that are generated from executing SQL*Loader. All files generated are denoted by the concurrent request ID plus the letter "l" representing line information.

Validations and Loading into the Open Interface Tables

The next step in the interface is to determine if there are any problems with the data and, if there are none, loading the data into the Payables Open Interface Tables. This is accomplished by executing a SQL*Plus session embedded in the UNIX shell program, which in turn calls a PL/SQL script. The core of the code is compiled in a custom package,

```
/*************************************/
/*** Set SQL*Loader Parameters ***/
/*************************************/
P_CTL="${DIRECTORY_SQLLDR}/empap_humvp_lines_tl.ctl"
P_DATA="${DIRECTORY_DATA}/${FILENAME_BASE}${BATCH_NUMBER}.ctl"
P_BAD="${DIRECTORY_BAD}/${REQUESTID}l.bad"
P_LOG="${DIRECTORY_BAD}/${REQUESTID}l.log"
P_DSC="${DIRECTORY_BAD}/${REQUESTID}l.dsc"

/*****************************************/
/*** Display SQL*Loader Parameters ***/
/*****************************************/
echo "Control file : ${P_CTL}"
echo "Log file   : ${P_LOG}"
echo "Data file  : ${P_DATA}"
echo "Bad file   : ${P_BAD}"
echo "Discard file : ${P_DSC}"

/**************************************/
/*** Execute SQL*Loader Session ***/
/**************************************/
sqlldr ${USERPASS} control=${P_CTL} data=${P_DATA} bad=${P_BAD} log=${P_LOG} discard=${P_DSC}

/*******************************************/
/*** Determine if there were problems ***/
/*******************************************/
ls ${P_BAD}
if [ $? -ne 0 ];
then
        echo "Data file loaded successfully into table empap_humvp_lines_tl."
else
        echo "ERROR: Problems occurred during data load into table empap_humvp_lines_tl."
        (cat ${P_BAD})|mailx -s "EMPAP: Invoice Upload Interface, Error with Concurrent
Request ${REQUESTID}" ${EMAILADDR1}
        exit 3
fi
```

Figure 14: UNIX Code for Lines Control File

ORAtips © 2006 Klee Associates, Inc.

which is executed from the PL/SQL program. The package validates the data, including the translation of the corporate COA combination into the spin-off COA combination, previously discussed. Figure 15 demonstrates how a PL/SQL program is executed from within a UNIX shell program.

The UNIX script not only executes the PL/SQL program, but also checks the status as to the success of data validation, loading the data into the Payables Open Interface tables, and the execution of Payables Open Interface Import. If all of these events were successful, UNIX sends a success e-mail to the caller, reporting the success along with the request ID for the concurrent request the caller submitted plus the request ID for the spawned concurrent request executed within PL/SQL code for Payables Open Interface Import. Simply put, a successful run of the Invoice Upload Interface will produce two concurrent requests: one for the interface

```
P_PROGRAM="${EMPCUST_TOP}/sql/empap_humvp_import.sql"

/**********************************/
/***  Execute PL/SQL Program  ***/
/**********************************/
sqlplus -s ${USERPASS} @${P_PROGRAM} ${USERNAME} ${ORGID} ${LOGINID}
${BATCH_NUMBER} ${RESPID} ${RESPAPPLID} ${SECGRPID}
ret_code=$?

/***********************************************************/
/***  Send Error Message if PL/SQL Program terminated  ***/
/***  in error.                                        ***/
/***********************************************************/
if test ${ret_code} -eq 1
then
        (echo "Exception error has occurred with request ${REQUESTID}, program EMPAP:
Invoice Upload Interface.")|mailx -s "EMPAP: Invoice Upload; Error, Request ID ${REQUESTID}"
${EMAILADDR1}
        exit 4

/***********************************************************/
/***  Send Error Message if there was a problem        ***/
/***  in loading the data to the Payables Open         ***/
/***  Interface tables.                                ***/
/***********************************************************/
elif test ${ret_code} -eq 2
then
        (echo "Problem with loading data into Payables Open Interface tables.  Please read
output for request ${REQUESTID}, program EMPAP: Invoice Upload Interface.")|mailx -s
"EMPAP: Invoice Upload; Error, Request ID ${REQUESTID}" ${EMAILADDR1}
        exit 5

/***********************************************************/
/***  Send Error Message if there was a problem in     ***/
/***  executing Payable Open Interface Import.         ***/
/***********************************************************/

elif test ${ret_code} -eq 3
then
        (echo "Unable to submit Payables Open Interface Import from request ${REQUESTID},
program EMPAP: Invoice Upload Interface.")|mailx -s "EMPAP: Invoice Upload; Error, Request ID
```

Figure 15: Executing PL/SQL Program - continued on next page

ORAtips © 2006 Klee Associates, Inc.

```
${REQUESTID}" ${EMAILADDR1}
            exit 6

/**********************************************************/
/***  Send Error Message if there was a problem in     ***/
/***  there was a data issue discovered during the     ***/
/***  execution of Payables Open Interface Import.     ***/
/**********************************************************/
elif test ${ret_code} -eq 4
then
            (echo "Problem with Payables Open Interface Import, program EMPAP: Invoice Upload
Interface.")|mailx -s "EMPAP: Invoice Upload; Error, Request ID ${REQUESTID}"
${EMAILADDR1}
            exit 7

/**********************************************************/
/***  Send Success Message if data validation was      ***/
/***  met and Payables Open Interface Import           ***/
/***  completed successfully.                          ***/
/**********************************************************/
else
            REQUESTID02=`echo "set heading off
             set echo off
             select request_id
             from empcustom.empap_humvp_header_tl
             where batch_number = '${BATCH_NUMBER}' and rownum = 1;" | sqlplus -s
${USERPASS}`
            REQUESTID02=`echo ${REQUESTID02} | sed 's;^ ;;g'`

            (echo "Payables Open Interface Import has been executed, request ${REQUESTID02},
program EMPAP: Invoice Upload Interface, request ${REQUESTID}.")|mailx -s "EMPAP: Invoice
Upload; Completed, Request ID ${REQUESTID}" ${EMAILADDR1}

fi
```

Figure 15: Executing PL/SQL Program - continued from previous page

and another for the Payable Open Interface Import, a child process of the interface.

So far I have only shown the execution of a concurrent program from within a UNIX shell program and discussed what general tasks are performed. Let us now look at the tasks more in-depth. The processing of data is too broad in scope to discuss in this article. Rather, I will list some highlights. Here are some of the validations, all performed within a PL/SQL package compiled on the database:

- Asset processing
- Validation of invoice number based on vendor
- Validation of vendor and site
- Determine if PO # (if provided) exists
- Validation of Humana COA combination
- Investigate accrued state tax, if provided
- Investigate accrued county tax, if provided
- Investigate accrued local tax, if provided
- Validation of payment term name
- Validation of invoice received date
- Validation of pay alone flag value
- Validation of special handle Flag

ORAtips © 2006 Klee Associates, Inc.

- Determine if payment method has been set up for vendor or site
- Asset location validation, if invoice is for an asset
- Asset major/minor category validation, if invoice is for an asset
- Asset tag # validation, if invoice is for an asset
- Asset expense account validation, if invoice is for an asset
- Determine if period is either open or future
- Determine if the line count is consistent with the control count
- Determine if the line amount is consistent with the control amount

Likewise, there are several tasks that must be performed before adding an invoice to the Open Interface tables. Invoices (not the Distributions) are added to a table called AP_INVOICES_INTERFACE. The data setup task required before records are added to this table, are the following:

- Retrieval of the respective vendor ID
- Retrieval of the respective vendor site ID
- Retrieval of the respective pay group Code
- Generation of a new invoice ID
- Retrieval of the terms ID
- Retrieval of the payment method for either the vendor or site

For each invoice there must be at least one distribution added to the Open Interface Tables. Invoice Distributions are added to a table called AP_INVOICE_LINES_INTERFACE. According to our business design, a distribution amount has to be entered into the Excel template. That value will be used alone for one distribution. The following is required before adding a distribution to the Open Interface Table:

- Translate the Humana COA combination to Emphesys COA combination
- Generate a new invoice line ID
- Determine the real line (distribution) count

A special note was made previously about the end users wanting to retain their original method for accounting (the Humana COA combination). This is the point where we take the original method of accounting and translate to the accounting method setup for Oracle Applications (Emphesys COA combination).

The final step is the actual execution of the concurrent program Payables Open Interface Import. Figure 16 gives a sample of the PL/SQL code that accomplishes this task.

```
/**********************************/
/***  Define Return Parameter  ***/
/**********************************/
variable ret_val number

/**********************************/
/***  Define Input Parameters  ***/
/**********************************/
define in_user_name = '&1'
define in_org_id = '&2'
define in_login_id = '&3'
define in_batch_num = '&4'
define in_resp_id = '&5'
define in_resp_appl_id = '&6'
define in_sec_grp_id = '&7'

/***********************************/
/***  Define Program Variables  ***/
/***********************************/
declare
        v_retcode           number;
        v_user_id apps.fnd_user.user_id%type;
        v_login_id          number;
```

Figure 16: Payables Open Interface Execution - Continued on next page

ORAtips © 2006 Klee Associates, Inc.

```
            v_batch_num         empcustom.empap_humvp_header_tl.batch_number%type;
            v_user_name         apps.fnd_user.user_name%type;
            v_org_id ap.ap_invoices_interface.org_id%type;
            v_req_id  number;
            v_import  boolean;
            v_period  empcustom.empap_humvp_header_tl.accounting_period%type;

begin
            :ret_val := 0;

/*******************************************************/
/***  Store Input Parameters into Program Variables  ***/
/*******************************************************/
            v_user_name := '&in_user_name';
            v_org_id := '&in_org_id';
            v_login_id := '&in_login_id';
            v_batch_num := '&in_batch_num';
            v_resp_id := '&in_resp_id';
            v_resp_appl_id := '&in_resp_appl_id';
            v_sec_grp_id := '&in_sec_grp_id';

/*********************************/
/***  Perform Data Validation  ***/
/*********************************/
            empcustom.empap_humvp_pk.process_excel_transactions_pr (
                        v_user_id
                        ,v_login_id
                        ,v_org_id
                        ,v_batch_num
                        ,v_resp_name
                        ,v_import
            );

/************************************************/
/***  Execute Payables Open Interface Import  ***/
/***  data passes validation.                 ***/
/************************************************/
            if v_import = true then
                        v_period := empcustom.empap_humvp_pk.get_period_fn(v_batch_num);
                        v_req_id := fnd_request.submit_request (
                                    'SQLAP',
                                    'APXIIMPT',
                                    'Payables Open Interface Import',
                                    null,
                                    false,
                                    'EMPAP_HUMVP',
                                    v_batch_num,
                                    v_batch_num,
                                    null,
                                    null,
                                    to_date('01-'||v_period, 'DD-MON-YY'),
                                    'Y',
                                    'N',
                                    'N',
                                    'N',
                                    1000,
```

Figure 16: Payables Open Interface Execution - Continued on next page

ORAtips © 2006 Klee Associates, Inc.

```
                              v_user_id,
                              v_login_id
                    );

/***************************************************/
/***  If the Request ID is 0, then there was  ***/
/***  a problem in submitting the request.    ***/
/***************************************************/
                    if v_req_id = 0 then
                              dbms_output.put_line (
                                        'Unable to submit Payables Open Interface Import.'
                              );
                              :ret_val := 3;

/*********************************************************/
/***  Else, the request was submitted successfully  ***/
/***  so record the Request ID for reporting.       ***/
/*********************************************************/
                    else

                              dbms_output.put_line (
                                        'Update batch for request id...'
                              );
                              update empcustom.empap_humvp_header_tl
                              set request_id = v_req_id
                              where batch_number = v_batch_num
                              ;
                              :ret_val := 0;
                    end if;

/*********************************************/
/***  Send back Validation Error Status  ***/
/*********************************************/
          else
                    :ret_val := 2;
          end if;
end;
/
exit :ret_val;
```

Figure 16: Payables Open Interface Execution - continued from previous pages

This PL/SQL script is slightly busy. First, the input parameters (from the UNIX shell script) are defined along with a return parameter that indicates the status for executing the PL/SQL script. All parameters are assigned to program variables. Validation is performed by a call to a custom package procedure: empcustom.empap_humvp_pk.process_excel_transactions_pr.

If the validation process has any errors, the return status indicator is set and returned to the UNIX shell program where, in turn, the concurrent program will be terminated in error. However, if all validation was successful, Payables Open Interface Import is executed. First, the period for the batch is retrieved via a call to the custom package function: empcustom.empap_humvp_pk.get_period_fn.

Next, Payables Open Interface Import is executed using the Oracle provided utility fnd_request.submit_request. If a request ID other than 0 is returned, the concurrent program was submitted successful. The program, in turn, records the child request ID for reporting purposes.

Let's take a closer look at the execution of Payables Open Interface Import. A normal way of executing this job from Oracle Applications is demonstrated in Figure 17.

You may notice that the source in Figure 17 differs from that in the

ORAtips © 2006 Klee Associates, Inc.

ORAtips

On Financials - Payables

source code in Figure 16. The source in Figure 16 is listed as "EMPAP_HUMVP", where in Figure 17, it is "Humana Upload". The reason is that the program in Oracle Applications accepts the user-friendly reference or "meaning" of the source. The "code" is used with fnd_request.submit_request. Since this is an interface, the source differs from those that are standard with Oracle Applications. Hence, the source must be set up. The source for loading invoices outside of Oracle Payables (such as through SQL*Loader) must be set up. This is accomplished by using the Oracle Payables Lookups window in Oracle Applications. Figure 18 demonstrates the source setup for this interface.

View of the Final Product

We have finally arrived at the final product of the interface. This is the easy part: demonstrating the execution of the interface. Depending on the responsibility, go into Oracle Applications and submit a concurrent request. In this case, our request will be a concurrent program called "EMPAP: Invoice Interface Upload". It requires only one parameter from the caller: the batch number. Provided that you have populated the Excel template and saved it as a comma-delimited file with proper naming convention, you will have a successful run of uploading an invoice batch to Oracle Payables. Figure 19 demonstrates submitting "EMPAP: Invoice Interface Upload".

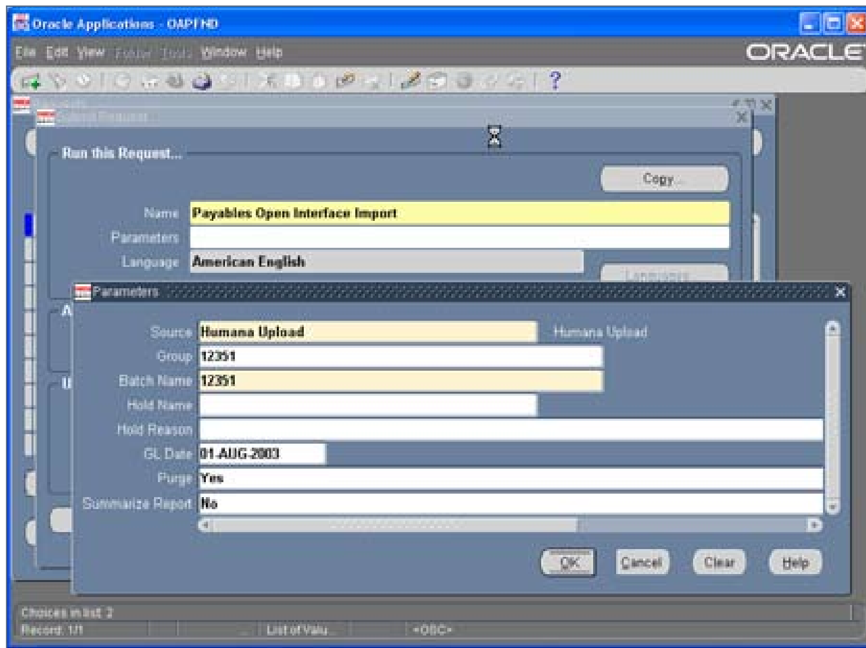

Figure 17: Payable Open Interface Import

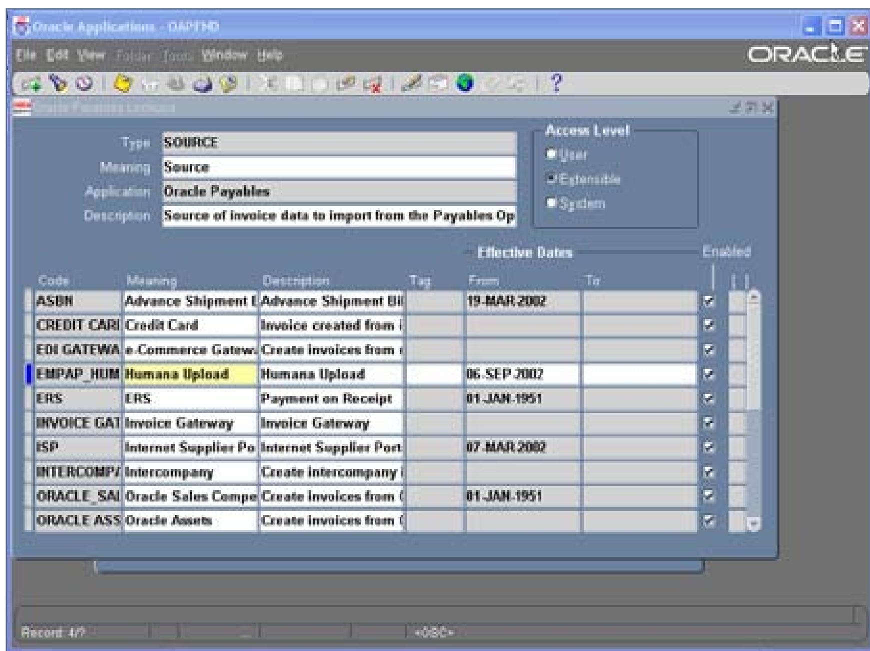

Figure 18: Oracle Payables Lookups Window

ORAtips © 2006 Klee Associates, Inc.

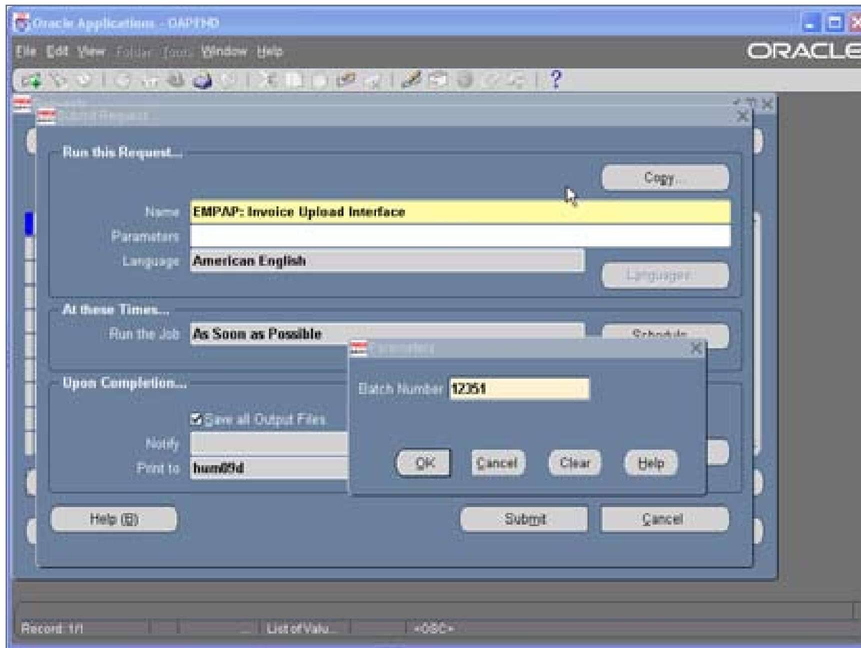

Figure 19: EMPAP: Invoice Upload Interface

Conclusion

As stated at the beginning, this is probably the busiest interface that has been deployed at Humana as part of Finance implementation of Oracle Payables. The interface has been a great success in processing. Given its power and performance, our associates are constantly reviewing the interface for enhancements. Stated briefly, it has been adapted to incorporate the process of mass additions via automation. Additionally, Humana incorporated the process of doing invoices for temporary vendors; or better stated, vendors that are invoiced only once. This incorporates the creation of vendors and sites prior to the creation of the invoices; a complicated process, yet extremely powerful tool. Also, enhancements have been provided for PO matching. The enhancements continue to this day. Given its success, I believe it provides other company benefits as well.

Kevin Ellis, *Humana* – Kevin has served as Technology / Applications Engineer for Humana at their headquarters in Louisville, KY since June 2000. His primary responsibility is to provide on-going support of Humana's Finance ERP (Oracle Applications 11.5.9). Additionally, he serves as the turn release manager for all enhancements released to the QA/Production environments. Kevin shares direct technical support with his staff for General Ledger, Payables, Fixed Assets, Purchasing, and Cash Management modules and writes SQL scripts for Discoverer workbooks, custom library, and forms. Additionally, Kevin (adjunct professor) teaches a graduate course in database theory at Bellarmine University (a private Catholic institution located in Louisville, KY). Kevin also teaches computer courses at Jefferson Community & Technical College. Kevin may be contacted at **Kevin.Ellis@ERPtips.com.**

ORAtips © 2006 Klee Associates, Inc.

ORAtips Journal

The information on our website and in our publications is the copyrighted work of Klee Associates, Inc. and is owned by Klee Associates, Inc. NO WARRANTY: This documentation is delivered as is, and Klee Associates, Inc. makes no warranty as to its accuracy or use. Any use of this documentation is at the risk of the user. Although we make every good faith effort to ensure accuracy, this document may include technical or other inaccuracies or typographical errors. Klee Associates, Inc. reserves the right to make changes without prior notice. NO AFFILIATION: Klee Associates, Inc. and this publication are not affiliated with or endorsed by Oracle Corporation. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Klee Associates, Inc. is a member of the Oracle Partner Network

This article was originally published by Klee Associates, Inc., publishers of JDEtips and SAPtips. For training, consulting, and articles on JD Edwards or SAP, please visit our websites: www.JDEtips.com and www.SAPtips.com.

ORAtips © 2006 Klee Associates, Inc.