ORAtips Journal October 2005 Volume I Issue 1

ORAtips
on Application Object Library

Real-Time Migration of Oracle® Application Setups Done In-House

By Kevin Ellis

Editor's Note: Migrating Application Object Library (AOL) objects amongst instances challenges many organizations. The process is time-consuming, error-prone, and manually intensive, draining administrators of valuable time, especially troubleshooting errors. Many third-party vendors have point products that provide COTS solutions, but these too can be time-consuming to integrate and costly. Kevin Ellis has developed a solution that uses in-house Oracle utilities (they come standard with your Oracle purchase) and simple SQL scripts to migrate AOL objects in real-time from one Oracle system to another and each with multiple instances.

Introduction

This paper will discuss the process of migrating objects (such concurrent programs) from one instance to another in real-time. This designed and developed under Oracle Application 11.5.9. However as part of the project, both instances will be upgraded to 11.5.10 before the actual migration takes place. In any event of the version the basic premise of migrating Application Object Library items is relatively the same.

Most companies have standard software deployment procedures. What that means in the Oracle world is that (at a minimum) there is an instance dedicated for developers, one for unit testing, one for quality assurance or integrated testing, and one for production. In the Oracle Application arena, one of the most common deliverables that goes through this cycle is that of Application Setups. An Application Setup (or an Application Object Library item) can be a concurrent program, request set, value set, descriptive flex field, and so on. It encompasses items that are set up on the front end or items set up via the screens of Oracle Applications.

.... your developers can deploy a system that can move one, two, or even a multitude of items from one instance to another in one sweep.

An Application Object Library (AOL) item is first set up manually in the development instance. Getting it from the development instance to the unit-testing instance is the next challenge. The same can be said for unit testing to QA and QA to production. You could buy something off the shelf that can perform this task for you, but that may require capital expenditures (for the software, consulting, and training of your staff) that may not be readily available. Another approach is to do it manually, but brings the risk that the AOL item gets set up incorrectly due to something as simple as a typo. It also is time consuming which indirectly costs the company money. And it is money that could have been invested toward something that is more productive. So, is there a way to move AOL items from one instance to another in real-time without doing it manually or investing money to buy a product off- the-shelf to do it? The answer to this question is yes.

This article will demonstrate how to move Application Object Library items (in real-time) from one instance to another. Your internal IT staff can accomplish these steps, assuming they are familiar with SQL coding and UNIX shell scripting. Database administration might also be necessary for a couple items as well, but that will be noted at the respective spots in the article. With these skill sets, your developers can deploy a system that can move one, two, or even a multitude of items from one instance to another in one sweep.

Background

One of the questions that I am often asked when I come up with a solution that can be applied to a multitude of problems is how I came up with the idea in the first place. In this case, it was a work related project that I was assigned to. The company that I work for uses Oracle Applications to manage Human Resources/Payroll and Finance data. However, they are segregated as two separate systems. On top of that, the Finance system is not the book of record. The actual book of record is a legacy system in which Oracle Applications (Finance) is a sub-ledger. As they say, with every question arises another question. This case is no different. Why

ORAtipsJournal October 2005 Volume I Issue 1

are there separate systems for Human Resources/Payroll and Finance? In fact, why are there two separate systems that maintain business data?

Let's start back at the beginning when I came on board, which was around June 2002. The company I work for (a health insurance provider) was planning to provide a health insurance product via the Web. This was to be the company's first attempt at Internet commerce. Additionally, it was thought that this new product might become the offspring of a spin off of the company. In other words, a subsidiary (or dot.com) company might be formed and would need its own set of books to manage finances. At the same time, the company was already converting their Human Resource/Payroll legacy system to Oracle Applications. Hence the potential dot-com company would use Oracle Applications to maintain Finance data.

At that time it was recognized that many spin-offs (especially those identified with the Internet) were falling out of favor. It was the time of the dot.com bust. The company gave a second thought to spinning off the new venture and decided to retain it in-house as part of their product development operations. Consequently, the setup of the Finance version of Oracle Applications would not be a book of record for a potential spin-off. Rather, it would be a sub-ledger to the company financial reporting system, which is legacy. Figure 1 displays this stage.

With three major systems to maintain, the company would like to convert the entire ledger legacy over to Oracle. Before it can get to that point, however, the two Oracle Application systems (Human Resources / Payroll & Finance) must first be merged. Hence, the project of merging the two Oracle Application systems is actually a sub-project leading to the much bigger goal of having a single system corporate wide that manages all aspects of Human Resources / Payroll and Finance data. Figure 2 demonstrates this stage.

Given this goal, comes our next question, "How do you migrate two Oracle Application systems into one?" In this case, the company decided to let the system of record be the Human Resources / Payroll system. That meant migrating UNIX architecture, security, transactional data, and setups from the Finance system to the Human Resources / Payroll system. This paper will focus on the last item: migrating setups (or Application Object Library items) from Finance to the Human Resources / Payroll system.

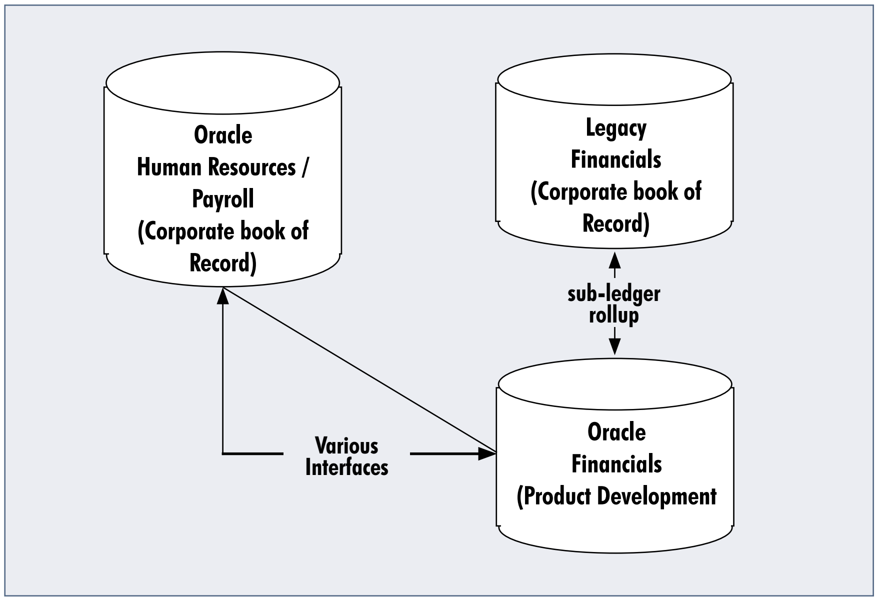
Oracle Human Resources / Payroll (Corporate book of Record)
Legacy Financials (Corporate book of Record)
sub-ledger rollup
Various Interfaces
Oracle Financials (Product Development

Figure 1 – No Spin-Off

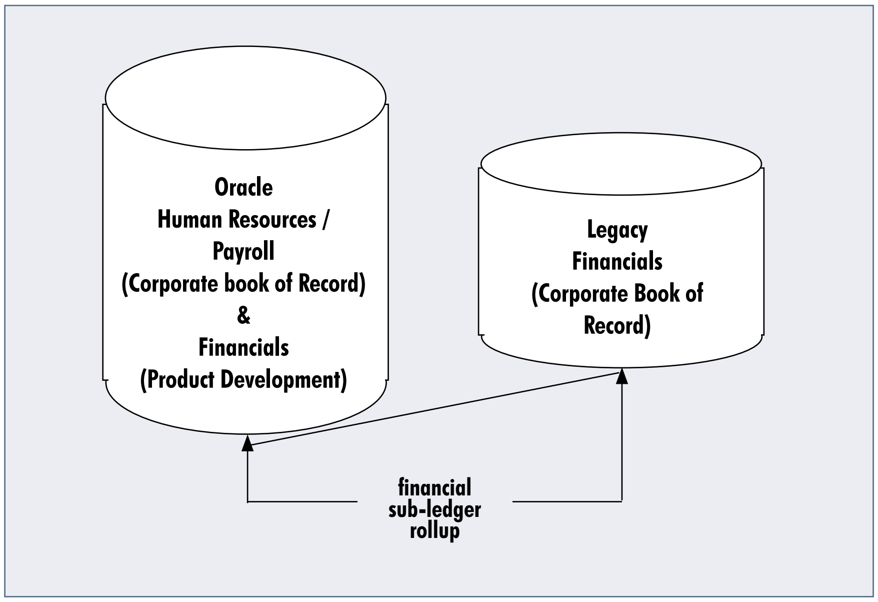
Oracle Human Resources / Payroll (Corporate book of Record) & Financials (Product Development)
Legacy Financials (Corporate Book of Record)
financial sub-ledger rollup

Figure 2 – Single Oracle System

The Model

The model is built around five basic steps. Those steps are the following:

1. Pre-Process Chain-of-Trust
2. Generation of the AOL Item List
3. Downloading the AOL Items
4. Uploading the AOL Items
5. Post-Process Chain-of-Trust

Figure 3 outlines the model from a top-level perspective.

Everything is executed from the HR/Payroll instance. The first task is to complete the Pre-Process of Chain of Trust. The next step is generating a list of AOL Items to be downloaded from the Finance system. Once this list is compiled, the next process that downloads all of the AOL Items from the Finance system can use it. Once all of the AOL items have been downloaded, they can then be uploaded to the HR / Payroll system. Once all AOL items have been uploaded, a final (post-process) Chain-of-Trust is preformed. Each of these processes will be explained in its entirety.

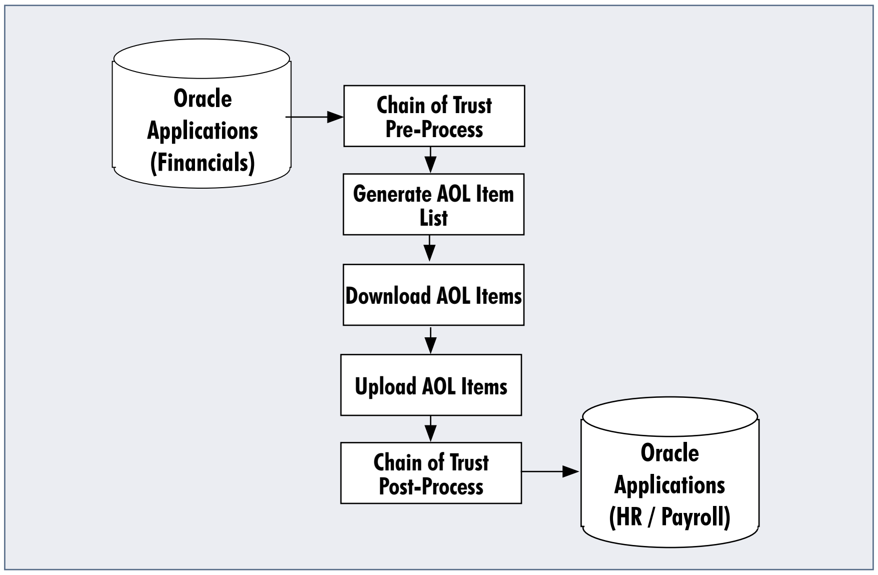
Oracle Applications (Financials)
Chain of Trust Pre-Process
Generate AOL Item List
Download AOL Items
Upload AOL Items
Chain of Trust Post-Process
Oracle Applications (HR / Payroll)

Figure 3 – AOL Migration Model

```
set pagesize 0
set heading off
set echo off
set verify off
set feedback off

select
        'Total Concurrent Programs - Destination'
from
        dual
;
select
        count(*)                                "Total"
from
        apps.fnd_concurrent_programs               a
        ,apps.fnd_concurrent_programs_tl        b
        ,apps.fnd_application                      c
where
        b.user_concurrent_program_name like 'EMP%'
        and a.concurrent_program_id = b.concurrent_program_id
        and a.application_id = c.application_id
        and a.enabled_flag = 'Y'
;

exit;
```

Figure 4 – Chain-of-Trust Count (Destination)

Chain-of-Trust (Pre-Process)

Chain-of-Trust is a verification procedure. Given that we know what must be migrated, we can generate a small collection of SQL scripts that can determine how many items from the Finance system will be migrated to the HR/Payroll system. Fortunately at the company I work for, we have naming standards for in-house or custom concurrent programs. In our case, every concurrent program starts with the prefix "EMP". (This was the acronym or first three letters for the name of the company that was to be spun off. Even though the company was not spun off, we retained the naming standard.) This proved very beneficial for us since custom concurrent programs in the HR/Payroll system began with the acronym "HUM", which were the first three letters of the parent company.

The Pre-Process Chain-of-Trust would search for concurrent programs in the Finance system and return a total for those whose names began with "EMP". Just to be on the safe side, the same thing would be done on the HR/Payroll side as well. Theoretically, there should be no concurrent programs in the HR/Payroll system that begin with "EMP", but I never assume anything. What should be returned is a positive number from the Finance system and 0 from the HR/Payroll system. Figure 4 lists the code used to determine the count of concurrent programs in the HR/Payroll system that begin with "EMP"

The same code is used to determine the count of concurrent programs that will be pulled from the Finance system. The exception is that this entire migration turn is done in real-time. Hence, it is executed from either the Finance or HR/Payroll instances. In our case, we execute all migration scripts from the destination system, which is HR/Payroll. Given this, a database link is required. Therefore, a separate copy of the SQL script is generated and edited to incorporate reference to the tables in the Finance system. The name of the database link must be attached to the table names in order to accomplish this. Figure 5 lists the code that accomplishes this task.

With this script, the link name is passed to the program. The migration scripts will always be executed from the destination system; hence, links are not required in the Chain-of-Trust Count (destination) SQL script. However, there will be several tests of the migration, which means it will be tested against many different instance and system names. Therefore, the link name will change for each test.

To conclude, the source script should return a positive count, whereas the destination script should be zero. These same two scripts will also be used as part of the Post-Process Train-of-Trust. In that case, the counts from both scripts (if everything was migrated successfully) should be zero. If there is a discrepancy, further analysis will be required. This will be covered under the section "Chain-of-Trust (Post-Process).

```
set heading off
set echo off
set verify off
set pagesize 0
set feedback off

define db_link='&1'

select
        'Total Concurrent Programs - Source'
from
        dual
;
select
        count(*)                            "Total"
from
        apps.fnd_concurrent_programs@&db_link              a
        ,apps.fnd_concurrent_programs_tl@&db_link     b
        ,apps.fnd_application@&db_link                     c
where
        b.user_concurrent_program_name like 'EMP%'
        and a.concurrent_program_id = b.concurrent_program_id
        and a.application_id = c.application_id
        and a.enabled_flag = 'Y'
;
exit;
```

Figure 5 – Chain-of-Trust Count (Source)

FNDLOAD (an Oracle-provided utility used for downloading/uploading AOL items) will be used.

Generate List of AOL Items

The next step in the migration process is generating a list of items to be migrated. In this case, an SQL script is deployed. The code is very similar to that used in the Chain-of-Trust scripts except it returns multiple records, not just a single value indicating the count of items to be downloaded. In this case, the list is a report of actual concurrent programs that will be downloaded from the Finance system and uploaded to the HR/Payroll system. FNDLOAD (an Oracle-provided utility used for downloading/uploading AOL items) will be used. The only information required would be the concurrent program name and the application short name to which the concurrent program is registered. The SQL script executes a SQL statement for this process and saves the results to a file (via spooling) to be used by the next processes. As mentioned previously, all code is executed from the destination system. Hence, a remote connection is required. Like that of the code in Figure 5, a database link is required for successful connection to the Finance system. Figure 6 lists the SQL code that accomplishes this task.

October 2005 Volume I Issue 1

ORAtipsJournal

```
set pagesize 0
set heading off
set echo off
set verify off
set feedback off

define db_link='&1'

spool conc_progs.txt
select
            a.concurrent_program_name
            ||' '||c.application_SHORT_name
from
            apps.fnd_concurrent_programs@&db_link                a
            ,apps.fnd_concurrent_programs_tl@&db_link      b
            ,apps.fnd_application@&db_link                       c
where
            b.user_concurrent_program_name like 'EMP%'
            and a.concurrent_program_id = b.concurrent_program_id
            and a.application_id = c.application_id
            and a.enabled_flag = 'Y'
;
spool off;
exit;
```

Figure 6 – Get AOL List

Download AOL Items from Source

Up to this point, all of the coding has been SQL scripts that are executed via SQL*Plus. You could use this SQL to do the actual download and import; however, it is much more difficult and in many cases Oracle will not support your system if you add customizations without going through the supported APIs provided. One type of API that Oracle provides is a UNIX utility called FNDLOAD. FNDLOAD is used extensively for patching. It is also the recommend utility for migrating AOL items from development, test, and production instances. Discussion on how to use FNDLOAD is a separate topic in itself. In this case study, we will focus on how it is used downloading concurrent programs from the Finance system and uploading them to the HR/Payroll system.

There are some general facts that must be recorded when downloading or uploading AOL items using FNDLOAD. First, each call of FNDLOAD generates a "log" file that records the success or failure of the download/update that just occurred. If FNDLOAD is used to download an AOL item, an additional file is created. This file is the AOL item setup. It has an "ldt" extension. The caller has control of what the actual file is named. Additionally, FNDLOAD is executed under the "APPS" account in Oracle. Hence, the "APPS" password is required for execution.

Our version of the migration involves downloading all of the custom concurrent programs in one full sweep. Hence, a UNIX script is generated that reads the AOL Item list generated from the code listed in Figure 6. As a result, a list of "ldt" and "log" files are generated. In order to keep all of these files in order and show what they relate to, a second list or report is generated. In this case, the report is called "conc_progs.txt". It looks very similar to the report "conc_progs.lst" created by the code in Figure 6, except it contains two additional fields: the name of the AOL Item setup file ("ldt") plus the "log" file generated from executing FNDLOAD. As a final step, all of the "ldt" setup files are zipped up. The UNIX code that performs this task is listed in Figure 7.

Oracle will not support your system if you add customizations without going through the supported APIs provided.

```
#*****************************
#*** Retrieve Parameters ***
#*****************************
P_APPS_PASSWORD=${1}
P_INSTANCE=${2}

#***************************************************
#*** Declaration of files used in this program ***
#***************************************************
P_ZIP_FILENAME="conc_progs"
P_CP_LIST="${P_ZIP_FILENAME}.lst"
P_CP_TEMP="${P_ZIP_FILENAME}.lst.temp"
P_CP_TEMP2="${P_ZIP_FILENAME}.lst.temp2"
P_CP_NAMES="${P_ZIP_FILENAME}.txt"
P_FND_PROGRAM="EMPXX_DOWNLOAD_CP.prog"
P_CP_LOG="LOG.txt"
P_CP_LOG_DIR="logs_cp"

rm ${P_CP_LIST}
rm ${P_ZIP_FILENAME}.zip

print "Creating Directory for LOG files..."
mkdir ${P_CP_LOG_DIR}
cd ${P_CP_LOG_DIR}
rm *.log
cd ..

echo " " > ${P_CP_LIST}
echo " " > ${P_CP_TEMP}
echo " " > ${P_CP_TEMP2}

P_LDT_FILES=""

#******************************************************************
#*** Start download of Concurrent Programs listed in .txt file ***
#******************************************************************
P_CP_SEQ="0"
P_CP_COUNT=`wc -l ${P_CP_NAMES} | awk '{print $1}'`

print " "
print "${P_CP_COUNT} Concurrent Programs to download."
print " "

while [ ${P_CP_COUNT} -ge 1 ]
do

#**************************************
#*** Increment LDT sequence number ***
#**************************************
        P_CP_SEQ=`expr ${P_CP_SEQ} + 1`
```

Figure 7 – UNIX Download Program – continued on next page

```
#***************************************************
#*** Retrieve Concurrent Program and     ***
#*** Application Short Name from .txt file ***
#***************************************************
        P_CP_SHORT_NAME=`cat ${P_CP_NAMES} | tail -${P_CP_COUNT} | head -1 | awk '{print $1}'`
        P_CP_APP=`cat ${P_CP_NAMES} | tail -${P_CP_COUNT} | head -1 | awk '{print $2}'`

#*****************************************************************
#*** Format the LDT filename for the Concurrent Program ***
#*** setup to download.                    ***
#*****************************************************************
        if [ ${P_CP_SEQ} -gt 9999 ]
        then
                P_CP_ISSUE="${P_CP_SEQ}"
        elif [ ${P_CP_SEQ} -gt 999 ]
        then
                P_CP_ISSUE="0${P_CP_SEQ}"
        elif [ ${P_CP_SEQ} -gt 99 ]
        then
                P_CP_ISSUE="00${P_CP_SEQ}"
        elif [ ${P_CP_SEQ} -gt 9 ]
        then
                P_CP_ISSUE="000${P_CP_SEQ}"
        else
                P_CP_ISSUE="0000${P_CP_SEQ}"
        fi
        P_CP_LDT="EMPXX${P_CP_ISSUE}"

        print "Generating ${P_CP_LDT}.ldt for ${P_CP_SHORT_NAME}, ${P_CP_APP} ..."

#*************************************************************************
#*** Execute FNDLOAD to download the specific Concurrent Program ***
#*************************************************************************
        C_FND_PATH="${FND_TOP}/patch/115/import"
        C_FND_PROGRAM="afcpprog.lct"
        P_FNDLOAD_PRG="${C_FND_PATH}/${C_FND_PROGRAM}"

        FNDLOAD apps/${P_CP_APP}@${P_INSTANCE} 0 Y DOWNLOAD ${P_FNDLOAD_PRG} ${P_CP_LDT}.ldt PROGRAM
        CONCURRENT_PROGRAM_NAME="${P_CP_SHORT_NAME}" APPLICATION_SHORT_NAME="${P_CP_APP}"

#*****************************************************************
#*** Record the new LDT setup file in the zip file list ***
#*****************************************************************
        P_LDT_FILES="${P_LDT_FILES} ${P_CP_LDT}.ldt"

#**********************************************************************
#*** Identify the LOG generated during the download and record ***
#*** it in the .lst file                ***
#**********************************************************************
```

Figure 7 – UNIX Download Program – continued on next page

October 2005 Volume I Issue 1

```
          ls *.log > ${P_CP_LOG}
          P_LOG_FILE_NAME=`cat ${P_CP_LOG} | awk '{print $1}'`
          P_RECORD="${P_CP_SHORT_NAME} ${P_CP_APP} ${P_CP_LDT} ${P_LOG_FILE_NAME}"
          echo "${P_RECORD}" > ${P_CP_TEMP}

          if [ ${P_CP_SEQ} -eq 1 ];
          then
                    cat ${P_CP_TEMP} > ${P_CP_LIST}
          else
                    cat ${P_CP_LIST} ${P_CP_TEMP} > ${P_CP_TEMP2}
                    cat ${P_CP_TEMP2} > ${P_CP_LIST}
          fi

#*****************************************
#*** Move LOG file to subdirectory ***
#*****************************************
          mv ${P_LOG_FILE_NAME} ${P_CP_LOG_DIR}

#*********************************
#*** Decrement loop counter ***
#*********************************
          print " "
          P_CP_COUNT=`expr ${P_CP_COUNT} - 1`

done

#*****************************************
#*** Put all LDT files in zip file ***
#*****************************************
print "Creating zip file ${P_ZIP_FILENAME} for Concurrent Program ldt files..."
zip ${P_ZIP_FILENAME} ${P_LDT_FILES}

#********************************
#*** Clean up staging area ***
#********************************
print " "
print "Remove ldt files..."
rm *.ldt
rm ${P_CP_LOG}
rm ${P_CP_TEMP}
rm ${P_CP_TEMP2}

print " "
print "End of Program."
#************************************
#*** Logical Ending of Program ***
#************************************
```

Figure 7 – UNIX Download Program

October 2005 Volume I Issue 1

Update AOL Items to Destination

This portion of the process is very similar to the downloading AOL items except that we are going in the opposite direction: uploading. We do not need redirection via a database link since all of the scripts are executed from the destination UNIX box. However, the process is similar in the fact that there is a primary UNIX script that will read a data file generated from the process of downloading AOL items and load them to the destination system via a call to FNDLOAD, which is wrapped in a separate UNIX script.

As with downloading AOL, uploading them using FNDLOAD generates a "log" file that indicates whether the records were successfully updated or not. FNDLOAD for uploading also requires the name of an existing "ldt" file. These filenames are part of the report file that is generated from the download process. Hence, all of the "ldt" files should exist. The only other parameter that is required is the "APPS" password.

Our version of the migration involves uploading all of the custom concurrent programs in one full sweep. Hence, a UNIX script is generated that reads the AOL Item list generated from the code listed in Figure 7. As a result, a list of "log" files is generated, different from the ones generated from the download. A separate report of these files is generated. It is identical to the one generated from the download except the "log" file names are different and pertain to information on uploading AOL. The report is called "conc_progs.upl" and looks very similar to the report "conc_progs.txt" created by the code in Figure 7. As a final step, all of the "ldt" setup files are zipped up. The UNIX code that performs this task is listed in Figure 8.

```
#******************************************
#*** Retrieve Parameters from caller ***
#******************************************
P_ZIP_FILENAME="conc_progs"
P_APPS_PASSWORD=${1}

#****************************************************
#*** Declaration of Files used in this program ***
#****************************************************
P_CP_LIST="${P_ZIP_FILENAME}.lst"
P_FND_LIST="${P_ZIP_FILENAME}.upl"
P_CP_TEMP="${P_ZIP_FILENAME}.lst.temp"
P_CP_TEMP2="${P_ZIP_FILENAME}.lst.temp2"
P_CP_NAMES="${P_ZIP_FILENAME}.txt"
P_FND_PROGRAM="EMPXX_UPLOAD.prog"
P_UPLOAD_LOG="LOG.txt"
P_CP_LOG_DIR="logs_upload"

#**************************************************
#*** Unzip the file containing all of the    ***
#*** LDT files that will be used for uploads ***
#**************************************************
print "Unzipping ${P_ZIP_FILENAME}.zip ..."
unzip ${P_ZIP_FILENAME}.zip

print "Creating Directory for LOG files..."
mkdir ${P_CP_LOG_DIR}
cd ${P_CP_LOG_DIR}
rm *.log
cd ..

echo " " > ${P_FND_LIST}
echo " " > ${P_CP_TEMP}
```

Figure 8 – UNIX Upload Program – continued on next page

```
echo " " > ${P_CP_TEMP2}

P_LDT_FILES=""

#*******************************************************************
#*** Process all setups stored in .lst filename provided ***
#*******************************************************************
P_FND_SEQ="0"
P_CP_COUNT=`wc -l ${P_CP_LIST} | awk '{print $1}'`

print " "
print "${P_CP_COUNT} Setups to upload."
print " "

while [ ${P_CP_COUNT} -ge 1 ]
do

#**********************************************************************
#*** Retrieve Short Name, Apps, and LDT filename parameters ***
#**********************************************************************
        P_FND_SEQ=`expr ${P_FND_SEQ} + 1`
        P_FND_SHORT_NAME=`cat ${P_CP_LIST} | tail -${P_CP_COUNT} | head -1 | awk '{print $1}'`
        P_FND_APP=`cat ${P_CP_LIST} | tail -${P_CP_COUNT} | head -1 | awk '{print $2}'`
        P_FND_LDT=`cat ${P_CP_LIST} | tail -${P_CP_COUNT} | head -1 | awk '{print $3}'`

        print "Uploading ${P_FND_LDT}.ldt for ${P_FND_SHORT_NAME}, ${P_FND_APP} ..."

#**************************************************************
#*** Execute FNDLOAD to upload concurrent programs ***
#**************************************************************
C_FND_PATH="${FND_TOP}/patch/115/import"
        C_FND_CONCURRENT_PROGRAM="afcpprog.lct"
        P_FNDLOAD_PRG="${C_FND_PATH}/${C_FND_CONCURRENT_PROGRAM }"

FNDLOAD apps/${P_APPS_PASSWORD} 0 Y UPLOAD ${P_FNDLOAD_PRG} ${P_FND_LDT}.ldt - CUSTOM_MODE=FORCE

#********************************************************
#*** Identify the LOG file that was generated ***
#*** for the most recent setup upload     ***
#********************************************************
        ls *.log > ${P_UPLOAD_LOG}
        P_LOG_FILE_NAME=`cat ${P_UPLOAD_LOG} | awk '{print $1}'`
        mv *.log ${P_CP_LOG_DIR}

#*********************************************************
#*** Record the Short Name, App, LDT filename, ***
#*** and LOG filename for the upload.      ***
#*********************************************************
        P_RECORD="${P_FND_SHORT_NAME} ${P_FND_APP} ${P_FND_LDT} ${P_LOG_FILE_NAME}"
        echo "${P_RECORD}" > ${P_CP_TEMP}

        if [ ${P_FND_SEQ} -eq 1 ];
```

Figure 8 – UNIX Upload Program – continued on next page

October 2005 Volume I Issue 1

```
            then
                        cat ${P_CP_TEMP} > ${P_FND_LIST}
            else
                        cat ${P_FND_LIST} ${P_CP_TEMP} > ${P_CP_TEMP2}
                        cat ${P_CP_TEMP2} > ${P_FND_LIST}
            fi

            print " "
            P_CP_COUNT=`expr ${P_CP_COUNT} - 1`

done

#**********************************
#*** Clear staging directory ***
#**********************************
print " "
print "Remove ldt files..."
rm *.ldt
rm ${P_CP_TEMP}
rm ${P_CP_TEMP2}
rm ${P_UPLOAD_LOG}

print " "
print "End of Program."
#************************************
#*** Logical Ending of Program ***
#************************************
```

Figure 8 – UNIX Upload Program

Chain-of-Trust (Post-Process)

As mentioned previously, Chain-of-Trust is a verification procedure. Like that of the "Pre-Process", the "Post-Process" verifies if all of the necessary AOL Items downloaded from the source system were migrated to the destination system. Given our current situation, this step determines if all of the concurrent programs downloaded from the Finance system were uploaded to the HR/Payroll system successfully.

The "Post-Process" uses some of the same scripts from the "Pre-Process". For instance, the code listed in Figure 5 is re-used to notify the user of how many concurrent programs were downloaded. The code in Figure 9 is used to identify discrepancies. Specifically, it lists the number of concurrent programs that were not migrated from Finance to HR/Payroll.

```
set echo off
set heading off
set verify off
set feedback off
set pagesize 0

define db_link='&1'

select
        '*******************************************************************************'
```

Figure 9 – Migration Verification (Count) – continued on next page

```
"Label"
from
        dual
;
select
        'Chain of Trust Verification - Differences'       "Label"
from
        dual
;
select
        count(*)                                          "Total"
from
        apps.fnd_concurrent_programs@&db_link                     a
        ,apps.fnd_concurrent_programs_tl@&db_link         b
        ,apps.fnd_application@&db_link                            c
where
        b.user_concurrent_program_name like 'EMP%'
        and a.concurrent_program_id = b.concurrent_program_id
        and a.application_id = c.application_id
        and a.enabled_flag = 'Y'
        and not exists (
                select
                        a.concurrent_program_name
                from
                        apps.fnd_concurrent_programs                      a1
                        ,apps.fnd_concurrent_programs_tl          b1
                        ,apps.fnd_application                             c1
                where
                        a1.concurrent_program_id =
                                b1.concurrent_program_id
                        and a1.application_id =
                                c1.application_id
                        and b1.user_concurrent_program_name =
                                b.user_concurrent_program_name
                        and c1.application_short_name =
                                c.application_short_name
        )
;
select
        '*************************************************************************************'
"Label"
from
        dual
;
exit;
```

Figure 9 – Migration Verification (Count)

The code listed in Figure 10 is very similar to that in Figure 9 except it specifically lists the missing concurrent programs that were not migrated from Finance to HR/Payroll.

The Driver Program

Part of the goal for the migration is to perform the necessary tasks in real-time. This is accomplished by writing a single driver program that calls all of the scripts listed above.

The code in Figure 11 is the main driver that will migrate all of the current programs from the Finance system to the HR/Payroll system in real-time.

```
set echo off
set heading off
set verify off
set feedback off
set pagesize 0

define db_link='&1'

select
        '******************************************************************************'"Label"
from
        dual
;
select
        'Missing Concurrent Programs'                       "Label"
from
        dual
;
select
        a.concurrent_program_name                   "Program"
        ,c.application_short_name                   "Application"
from
        apps.fnd_concurrent_programs@&db_link               a
        ,apps.fnd_concurrent_programs_tl@&db_link   b
        ,apps.fnd_application@&db_link                      c
where
        b.user_concurrent_program_name like 'EMP%'
        and a.concurrent_program_id = b.concurrent_program_id
        and a.application_id = c.application_id
        and a.enabled_flag = 'Y'
        and not exists (
                select
                        a.concurrent_program_name
                from
                        apps.fnd_concurrent_programs                a1
                        ,apps.fnd_concurrent_programs_tl    b1
                        ,apps.fnd_application                       c1
                where
                        a1.concurrent_program_id =
                                b1.concurrent_program_id
                        and a1.application_id =
                                c1.application_id
                        and b1.user_concurrent_program_name =
                                b.user_concurrent_program_name
                        and c1.application_short_name =
                                c.application_short_name
        )
;
select
        '******************************************************************************' "Label"
from
        dual
;
exit;
```

Figure 10 – Migration Verification (Summary)

ORAtips Journal October 2005 Volume I Issue 1

The driver is executed from the UNIX shell prompt. It requires five parameters:

- the "APPS" schema password to the destination system
- the password to your "custom" schema (given that your customizations have been compiled to a custom schema)
- for the destination system the remote DB Link name
- the "APPS" password to the source system
- the password to the custom schema in the source system

Conclusion

This article has demonstrated how to migrate concurrent programs from one Oracle instance to another. In this case, it was used to migrate two separate systems into one. However, it can be used as an enhancement turn process, such as turn concurrent programs from development to unit testing to quality assurance testing to production. Even though the example demonstrated was with concurrent programs, the same model can be used for request sets, descriptive flex fields, values sets, profiles, and many more varieties of AOL Items. It all can be done in real-time. The key to successful implementation is analysis: what do you want to migrate. Careful planning can help you avoid pitfalls or bottlenecks that may arise.

Kevin Ellis, *Humana* – Kevin has served as Technology / Applications Engineer for Humana at their headquarters in Louisville, KY since June 2002. 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 may be contacted at **Kevin.Ellis@ERPtips.com**

```
#***********************************
#*** Declaration of Constants ***
#***********************************
P_LINE_SUB="==================================================================================="
P_DIR=`pwd`
P_CONN_APP="apps/"${1}
P_CONN_EMP="custom/"${2}
P_DBLINK=${3}
P_SRC_APPS_PASS="apps/"${4}
P_SRC_INSTANCE=${5}

#*********************************
#*** Declaration of Scripts ***
#*********************************
P_PROG001="cp_get_list.sql"
P_PROG002="cp_count.sql"
P_PROG003="cp_count_remote.sql"
P_PROG004="CONV_DOWNLOAD_CP.prog"
P_PROG005="CONV_UPLOAD_CP.prog"
P_PROG006="cp_verify.sql"
P_PROG007="cp_missing.sql"

#***************************************
#*** Chain-of-Trust (Pre-Process) ***
#***************************************
```

Figure 11 – The Driver – continued on next page

```
P_CUR_DIR=`pwd`
print "Get Count of Concurrent Programs in Destination."
sqlplus -s ${P_CONN_APP} @${P_CUR_DIR}/${P_PROG002}
print " "
print "Get Count of Concurrent Programs in Source."
sqlplus -s ${P_CONN_APP} @${P_CUR_DIR}/${P_PROG003} ${P_DBLINK}
print " "

#*************************************
#*** Generate List of AOL Items ***
#*************************************
print "Get list of Concurrent Programs to download"
sqlplus -s ${P_CONN_APP} @${P_CUR_DIR}/${P_PROG001} ${P_DBLINK}

#*****************************************
#*** Download AOL Items from Source ***
#*****************************************
print "Download Concurrent Programs .."
${P_PROG004} ${P_SRC_APPS_PASS} ${P_SRC_INSTANCE}

#******************************************
#*** Upload AOL Items to Destination ***
#******************************************
print "Upload Concurrent Programs .."
${P_PROG005} ${P_CONN_APP}

#***************************************
#*** Chain-of-Trust (Post-Process) ***
#***************************************
P_CUR_DIR=`pwd`
print "Verify Concurrent Programs were uploaded."
sqlplus -s ${P_CONN_APP} @${P_CUR_DIR}/${P_PROG002}
print " "

sqlplus -s ${P_CONN_APP} @${P_CUR_DIR}/${P_PROG006} ${P_DBLINK}
print " "

sqlplus -s ${P_CONN_APP} @${P_CUR_DIR}/${P_PROG007} ${P_DBLINK}
print " "
#***********************************************
#*** Logical Ending of Issue Patch Script ***
#***********************************************
```

Figure 11 – The Driver

ORAtips Journal