

Repeatable, Reliable – FNDLOAD

Susan Behn, *Solution Beacon, LLC*
Gerald Jones, *Solution Beacon, LLC*

Introduction

Migration of setups and data between instances falls into two basic business needs. The first business need is to easily migrate data and setups during the implementation process between development, the various testing instances and ultimately production. In this case, typically all data including setup data and non transactional application data such as suppliers, bank accounts, inventory organizations, chart of accounts, calendars, etc... needs to be easily migrated. The second need is on-going change management. On-going change management is typically limited to setup data and often involves technical components such as concurrent program definitions for custom reports and extensions, personalizations, new printers, new menus, etc... Different tools are more efficient for each of these two basic needs. In this paper, we will focus on the change management component for which FNDLOAD is a good migration option.

Overview of FNDLOAD

FNDLOAD is a developer tool provided by Oracle that migrates data between Oracle Application instances. It is executed at the UNIX command line or via unix scripts created by the developer. Oracle provides configuration files for AOL setup data, HR setups and AME. These configuration files define the parent and child entities to be downloaded and uploaded. This “out of the box” standard functionality ensures a reliable, repeatable process to migrate setup data between instances.

The download process creates text based data files that are transferable to any Oracle Applications instance. As long as there are no table changes between the release levels, it is not critical for the source and target instance to be at the exact same patch level in most cases. Data preservation options to be discussed later in this paper will protect the target data from inappropriate update based on the owner and/or last update date. This makes FNDLOAD ideal as a change management tool.

Other Migration Alternatives

The most often utilized alternative sadly is manual entry. This option clearly results in data entry errors, differences between instances and is the time consuming. No one types the same thing twice the same way every time. There is significant frustration when a setup is implemented in production with a deviation from the source instance causing a different behavior than what was tested in other environments. Any of the other alternatives are preferred over manual entry.

Several third party applications are available to move setup and application data between instances. These typically include audit features as well. While these options are often clearly the Cadillac of migration options, they tend to be expensive and require training and maintenance. For those businesses with significant security and audit requirements, some of these applications are worth considering. For this discussion, we will not venture into the many third party options available.

iSetup is the nearest Oracle provided alternative to FNDLOAD that will meet some of the needs for change management. iSetup is ideal during implementations and is a front end functional tool. iSetup additionally has the advantage of being able to migrate application data where FNDLOAD is only for setup data. However, several significant limitations are a factor in the change management process. iSetup does not consider the owner or timestamp in order to preserve the seeded data or most recent version of data. iSetup migrates data only in the primary language. There are no multi-language capabilities. More significantly for change management, iSetup has limited ability to migrate specific objects. In some cases, filters are available to migrate individual setups such as a single concurrent

program, but in many cases, it's all or nothing. The source and target instance must be at the same patch level for iSetup. iSetup is not an option during the upgrade process even for a point release upgrade. Finally, iSetup will not migrate Approvals Management entities.

What can be Migrated Between Instances

As stated earlier, FNDLOAD is used to migrate configuration data. The chart below includes a sample of the many of the entities that can be migrated with FNDLOAD. This is not an all inclusive list.

Request Groups, Request Sets	User Responsibilities
Profile Options	Printer Definitions
Key and Descriptive Flexfields	FND Dictionary
Menus and Responsibilities	Help Configuration
Forms and Form Functions	Document Sequences
Attachments	Concurrent Manager Schedules
Messages	Forms Personalizations
Value Sets and Values	Approvals Management Objects ***Not available in iSetup

In contrast to FNDLOAD, the following additional items can be migrated in iSETUP.

Discrete Mfg and Distribution Setups	Payroll Elements
Employees	Product Foundation
Financials Operating Units	Profitability Manager Setups
Financials Setups	Suppliers
GL Daily Rates	Transfer Pricing Setups
Organization Structure	XML Publisher

Data Preservation

FNDLOAD will preserve data in the target instance based on the following rules:

1. If the OWNER equals SEED in the source instance, then records from the source instance will never overwrite records in the target instance where the OWNER equals CUSTOM.
2. If the OWNER equals CUSTOM in the source instance, then records from the source instance will always overwrite records in the target instance where the OWNER equals SEED.
3. If the OWNER is the same in both instances, than the record with the most recent LAST_UPDATE_DATE will be placed or retained in the target instance

This is one of the significant differences between iSetup and FNDLOAD.

How to Migrate Data

The executable for FNDLOAD is located in \$FND_TOP/bin. FNDLOAD can be executed from the command line with the following parameters or ideally placed in a UNIX script that logs successful uploads or downloads. The syntax of FNDLOAD is as follows:

```
FNDLOAD <apps/$APPS_PW> 0 Y <Mode> <Configfile> <target data file> <entity> <parameter>
```

- **0 Y** = Concurrent Program Flags
- **Mode** = UPLOAD, UPLOAD_PARTIAL or DOWNLOAD
- **Configfile** = Configuration file (.lct) provided by Oracle in \$FND_TOP/patch/115/import

- **Target Data File** = Name of the file (.ldt) to be created by Download or used by Upload. This file contains the definition of the entity being migrated.
- **Entity** = type of object being migrated (printer style, lookup, executable,...)
- **Parameter** = parameter related to the entity (Which printer style, lookup, etc)

An example of a script to download concurrent programs is shown below. This script creates a download log which will retain any errors that occur during the execution of FNDLOAD.

```
#!/bin/ksh
#*****

cpgm_file_name=$1

# source password file
. $APPL_TOP/.applenv

apps_db_passwd=$APPS_PASS
#app_short_name=SLCUST1
log_file=$PWD/FNDLOAD.$cpgm_file_name.download.log
fndload_file=FNDLOAD_$cpgm_file_name.ldt

echo 'Downloading Concurrent Program Details for Concurrent Program Short Name' $cpgm_file_name 'to
file' $PWD/FNDLOAD_$cpgm_file_name.ldt >> $log_file

FNDLOAD apps/$apps_db_passwd O Y DOWNLOAD $FND_TOP/patch/115/import/afcpprog.lct $fndload_file PROGRAM
APPLICATION_SHORT_NAME="XXSB" CONCURRENT_PROGRAM_NAME=$cpgm_file_name 2>/dev/null

grep "DEFINE PROGRAM" $fndload_file >> $log_file

if [ $? != 0 ] ; then
    echo FNDLOAD of Concurrent Program for concurrent program short name $cpgm_file_name was not
    successful >> $log_file

    exit 1
else
    echo FNDLOAD of Concurrent Program for concurrent program short name $cpgm_file_name was successful
    >> $log_file
fi;

exit 0
```

When downloading data, there is a choice to download all data for the entity or use the parameters available to limit downloaded data. If all data is downloaded, the parameters can be used at the upload stage by using the UPLOAD_PARTIAL mode. Examples are shown below.

To download a single concurrent program identified with the executable name GLIMP, use the following command:

```
FNDLOAD apps/apps 0 Y DOWNLOAD $FND_TOP/patch/115/import/afcpprog.lct mydata.ldt
CONCURRENT_PROGRAM CONCURRENT_PROGRAM_NAME='GLIMP'
```

If the data is limited in the download, there is no need to limit data on the upload. To upload all data that exists in the mydata.ldt file, use the following command:

```
FNDLOAD apps/apps 0 Y UPLOAD $FND_TOP/patch/115/import/afcpprog.lct mydata.ldt
```

To download all concurrent program data with only the limitation of the application name, use the following command:

```
FNDLOAD apps/apps 0 Y DOWNLOAD $FND_TOP/patch/115/import/afcpprog.lct mydata.ldt
APPLICATION_NAME='FND'
```

In this case, you may want to only migrate some of the concurrent programs to the target instance. Use UPLOAD_PARTIAL mode as shown the following command to upload only some of the data that exists in the mydata.ldt file:

```
FNDLOAD apps/apps 0 Y UPLOAD_PARTIAL $FND_TOP/patch/115/import/afcpprog.lct  
mydata.ldt CONCURRENT_PROGRAM CONCURRENT_PROGRAM_NAME='GLIMP'
```

Configuration Files

Configurations files are provided by Oracle for AOL, HR and AME entities. AOL configuration files are located in \$FND_TOP/patch/115/import. HR and AME configuration files are located in \$PER_TOP/patch/115/import in Release 11*i*. In release 12, the AME configuration files have moved to \$AME_TOP/patch/12/import. These configuration files have four main sections. The comments at the beginning of the configuration files often identify the parent and child entities included in the load process as well as parameters available to limit the data. However, the comments are not always consistent. Fortunately, the other sections are fairly intuitive and you should have no problem identifying the entities or the parameters. The Define Block section specifies the structure of the entities to be migrated. It defines key attributes, base attributes, trans (translation) attributes, CTX (context) attributes and child entities. The Download Block includes the SQL statements to download the entities. Look for the bind variables to identify available parameters. The Upload Block is a SQL statement or anonymous PL/SQL block to upload the data to the target instance. Appendix A in this document shows a marked up example of the configuration file for forms personalizations highlighting the various sections and key components.

Approvals Management (AME) Examples

The process for FNDLOAD is the same for all entities as shown in the earlier example for concurrent programs. The only difference is the parameters and the name of the configuration file. Since Approvals Management (AME) is new and documentation is limited, we have chosen to provide the detailed examples for these entities. The FNDLOAD commands for AME were recently tested in an 11.5.10.2 vision database and appear to function correctly as designed; however, we encourage extensive testing particularly for these entities since they were recently added.

The following AME setup components can be migrated with FNDLOAD:

- Transaction Types
- Approver Types
- Item Classes
- Attributes
- Conditions
- Action Type Configurations
- Approver Groups
- Rules

The following lists the commands necessary to download the setup data from the database into a flat file. Subsequently listed are the commands necessary to upload the data from the flat file to the database. Also included are notes about effective use of the commands or setup recommendations.

Transaction Types (Calling Apps)

Download

FNDLOAD apps/<apps pwd> 0 Y DOWNLOAD amescvar.lct <download file name>.ldt
AME_CALLING_APPS APPLICATION_SHORT_NAME=<FND application short name>
TRANSACTION_TYPE_ID=<AME transaction type short name>

Ex. FNDLOAD apps/apps 0 Y DOWNLOAD amescvar.lct sbtrantype.ldt
AME_CALLING_APPS APPLICATION_SHORT_NAME=SQLAP
TRANSACTION_TYPE_ID=SBTRANSTYPE

Upload

FNDLOAD apps/<apps pwd> 0 Y UPLOAD amescvar.lct <download file name>.ldt

Ex. FNDLOAD apps/apps 0 Y UPLOAD amescvar.lct sbtrantype.ldt.ldt

Attributes

Download of attributes requires two different scripts to be executed. One script downloads the attribute structure itself and the other downloads the data contained in the value field of the attribute. For example the SQL query in a dynamic attribute field. Both scripts allow you to download all of the current attributes for a give AME transaction type. Additionally you can download a single attribute or multiple attributes using a pattern string (e.g. SB%). It is recommended that if you are creating attributes for an existing transaction type, name any new or custom attributes starting with the same prefix. This will allow for them to be singled out and downloaded as a group at migration time. If creating a new transaction type, download of all attributes using only the application short name and transaction type parameters.

Download

Attributes

FNDLOAD apps/<apps pwd> 0 Y DOWNLOAD amesmatt.lct <download file name>.ldt
AME_ATTRIBUTES APPLICATION_SHORT_NAME=<FND application short name>
TRANSACTION_TYPE_ID=<AME transaction type short name>
[ATTRIBUTE_NAME=<attribute_name>]

Ex. FNDLOAD apps/apps 0 Y DOWNLOAD amesmatt.lct sbattributes.ldt
AME_ATTRIBUTES APPLICATION_SHORT_NAME=SQLAP
TRANSACTION_TYPE_ID= SBTRANSTYPE
ATTRIBUTE_NAME=SB_CUST_ATTRIBUTE

Attribute Usages

FNDLOAD apps/<apps pwd> 0 Y DOWNLOAD amesmatr.lct <download file name>.ldt
AME_ATTRIBUTE_USAGES APPLICATION_SHORT_NAME=<FND application short name>
TRANSACTION_TYPE_ID=<AME transaction type short name>
[ATTRIBUTE_NAME=<attribute_name>]

Ex. FNDLOAD apps/apps 0 Y DOWNLOAD amesmatr.lct sbattribute_usages.ldt
AME_ATTRIBUTE_USAGES APPLICATION_SHORT_NAME=SQLAP

**TRANSACTION_TYPE_ID=SBTRANSTYPE
ATTRIBUTE_NAME=SB_CUST_ATTRIBUTE**

Upload

Attributes

FNDLOAD apps/<apps pwd> 0 Y UPLOAD amesmatt.lct <download file name>.ldt

Ex. FNDLOAD apps/apps 0 Y UPLOAD amesmatt.lct sbattributes.ldt

Attribute Usages

FNDLOAD apps/<apps pwd> 0 Y UPLOAD amesmatr.lct <download file name>.ldt

Ex. FNDLOAD apps/apps 0 Y UPLOAD amesmatr.lct sbattribute_usages.ldt

Conditions

The script that downloads AME conditions allows you to download all conditions for a given transaction type or only those associated with a particular attribute or group of attributes. As mentioned in the attributes section, it is recommended that if you are creating conditions on attributes for an existing transaction type, name any new or custom attributes starting with the same prefix. This will allow for the associated conditions to be singled out and downloaded as a group at migration time. If creating a new transaction type, download of all conditions using only the application short name and transaction type parameters.

Download

FNDLOAD apps/<apps pwd> 0 Y DOWNLOAD amesconk.lct <download file name>.ldt
AME_CONDITIONS APPLICATION_SHORT_NAME=<FND application short name>
TRANSACTION_TYPE_ID=<AME transaction type short name>
[ATTRIBUTE_NAME=<attribute_name>]

**Ex. FNDLOAD apps/apps 0 Y DOWNLOAD amesconk.lct sbconditions.ldt
AME_CONDITIONS APPLICATION_SHORT_NAME=SQLAP
TRANSACTION_TYPE_ID=SBTRANSTYPE**

Upload

FNDLOAD apps/<apps pwd> 0 Y UPLOAD amesconk.lct <download file name>.ldt

Ex. FNDLOAD apps/apps 0 Y UPLOAD amesconk.lct sbconditions.ldt

Approval Groups

There are two scripts that are required to be executed to migrate custom approver groups. One script downloads the structure of the approver group including the SQL query used to select members for any

dynamic queries. The other script downloads data regarding the order number and voting regime of the approver group. There are several notes of interest regarding downloading of approver groups.

- The download script only works for dynamic approver groups. It does not work for static approver groups.
- The script does not readily recognize approver group names containing spaces. If the approver group(s) contains spaces, add the % wildcard symbol in between each word of the approver group to ensure it is downloaded properly
- The primary approver group script does not allow for downloading all approver groups for a given transaction type. You must supply the name of an approver group or some matching pattern of multiple groups if the naming convention of the groups is similar.
- To add to the previous point, it is recommended that in order to be able to download multiple approver groups at one time (instead of having to create multiple download files), use a common prefix when naming the approver group. For example, use <application short name>_cust as the prefix for approver group such as AP cust (name of the approver group).
- The approver group configuration script does allow all approver group configuration for a given transaction type to be downloaded at one time.

Download

Approval Group

```
FNDLOAD apps/<apps pw> 0 Y DOWNLOAD amesappg.lct <download file name>.ldt  
AME_APPROVAL_GROUPS APPROVAL_GROUP_NAME=<Approval Group Name>
```

#Need to use wildcards if Approval Group name has spaces

```
Ex. FNDLOAD apps/apps 0 Y DOWNLOAD amesappg.lct sbappgrps.ldt  
AME_APPROVAL_GROUPS APPROVAL_GROUP_NAME=SB %Cust %Grp %
```

Approval Group Configuration

```
FNDLOAD apps/<apps pw> 0 Y DOWNLOAD amesaagc.lct <download file name>.ldt  
AME_APPROVAL_GROUP_CONFIG APPLICATION_SHORT_NAME=<FND application  
short name> TRANSACTION_TYPE_ID=<AME transaction type short name>  
[APPROVAL_GROUP_NAME=<Approval Group Name>]
```

#Need to use wildcards if Approval Group name has spaces

```
Ex. FNDLOAD apps/apps 0 Y DOWNLOAD amesaagc.lct sbappgrpscon.ldt  
AME_APPROVAL_GROUP_CONFIG APPLICATION_SHORT_NAME=SQLAP  
TRANSACTION_TYPE_ID=SBTRANSTYPE  
APPROVAL_GROUP_NAME=SB %Cust %Grp %
```

Upload

Approval Group

```
FNDLOAD apps/<apps pw> 0 Y UPLOAD amesappg.lct <download file name>.ldt
```

Ex. FNDLOAD apps/apps 0 Y UPLOAD amesappg.lct sbappgrps.ldt.ldt
Approval Group Configuration

FNDLOAD apps/<apps pw> 0 Y UPLOAD amesaagc.lct <download file name>.ldt

Ex. FNDLOAD apps/apps 0 Y UPLOAD amesaagc.lct sbappgrpscon.ldt

Action Type Configurations

Action Type configurations refers to the download of action types that have been enabled for a given transaction type. In other words, you can migrate the action types that have been enabled and configured for a transaction type from one instance to another. The download does allow all configurations for a given transaction type to be downloaded at one type using the application short name and transaction type parameters of the script. You can also download by specific action type or a group of action types using a specific string pattern to match against.

Download

FNDLOAD apps/<apps pw> 0 Y DOWNLOAD amesaatc.lct <download file name>.ldt
AME_ACTION_TYPE_CONFIG APPLICATION_SHORT_NAME=<FND application short name>
TRANSACTION_TYPE_ID=<AME transaction type short name>
[ACTION_TYPE_NAME=<action type name>]

#Need to use wildcards if Action Type name has spaces

Ex. FNDLOAD apps/apps 0 Y DOWNLOAD amesaatc.lct sbacttconf.ldt
AME_ACTION_TYPE_CONFIG APPLICATION_SHORT_NAME=SQLAP
TRANSACTION_TYPE_ID=SBTRANSTYPE

Upload

FNDLOAD apps/<apps pw> 0 Y UPLOAD amesaatc.lct <download file name>.ldt

Ex. FNDLOAD apps/apps 0 Y UPLOAD amesaatc.lct sbacttconf.ldt

Rules

The download of AME rules requires two scripts to be executed. The first script downloads information about the rule (e.g. name, description, etc) along with associated conditions and rule type. The second script downloads all associated actions for the rule. Both scripts allow all rules for a given transaction type to be downloaded. You can also download a specific rule. However, unlike some of the previous components, you cannot download a group of rules using a string wildcards. The reason for this is that the FNDLOAD scripts use the Oracle generated rule key as a parameter to download a specific rule. It is recommended to download all rules for a transaction type unless only a handful of new rules have been created and need to be migrated.

Download

Rules

FNDLOAD apps/<apps pw> 0 Y DOWNLOAD amesrulk.lct <download file name>.ldt
AME_RULES APPLICATION_SHORT_NAME=<FND application short name>
TRANSACTION_TYPE_ID=<AME transaction type short name> [RULE_KEY=<Rule Key>]

Rule Key is found in AME_RULES table

**Ex. FNDLOAD apps/apps 0 Y DOWNLOAD amesrulk.lct sbrules.ldt AME_RULES
APPLICATION_SHORT_NAME=SQLAP TRANSACTION_TYPE_ID=SBTRANSTYPE**

Rule Actions

FNDLOAD apps/<apps pw> 0 Y DOWNLOAD amesactu.lct <download file name>.ldt
AME_ACTION_USAGES APPLICATION_SHORT_NAME=<FND application short name>
TRANSACTION_TYPE_ID=<AME transaction type short name> [RULE_KEY=<Rule Key>]

Rule Key is found in AME_RULES table

**Ex. FNDLOAD apps/apps 0 Y DOWNLOAD amesactu.lct sbrulesact.ldt
AME_ACTION_USAGES APPLICATION_SHORT_NAME=SQLAP
TRANSACTION_TYPE_ID=SBTRANSTYPE**

Upload

Rules

FNDLOAD apps/<apps pw> 0 Y UPLOAD amesrulk.lct <download file name>.ldt

Ex. FNDLOAD apps/apps 0 Y UPLOAD amesrulk.lct sbrules.ldt

Rule Actions

FNDLOAD apps/<apps pw> 0 Y UPLOAD amesactu.lct <download file name>.ldt

FNDLOAD apps/apps 0 Y UPLOAD amesactu.lct sbrulesact.ldt

Migration Issues

FNDLOAD does not always accommodate the deletion of rows. For example, FNDLOAD will not delete stages or programs from request sets when uploading. However, FNDLOAD will delete parameters in a concurrent program definition. Check the configuration file to see if any data is deleted as part of the upload process. It may be necessary to create a custom configuration file if this is a problem, but keep in mind that Oracle does not support custom configuration files.

Another issue, particularly with the AME configuration scripts has to do with updates to existing data. For example, AME scripts work well when downloading and migrating setup data to another instance that does not currently contain an instance of the data (e.g. attribute). However, if existing data is updated in the source instance and subsequently downloaded using FNDLOAD and uploaded again to the same instance, the scripts do not update existing setup components well, if at all.

XDO Loader

Although this is out of scope for this particular white paper, it is worth mentioning that XDOLoader is a java-based utility to load template files for XML publisher similar to FNDLOAD. Refer to MetaLink Document Id: 469585.1 for details on how to use XDOLoader.

Conclusion

It was the intent of this paper to provide an overview of FNDLOAD to encourage the reader to use automated tools to migrate setups and configuration data over manual methods. FNDLOAD is certainly not the only option. Consider the pros and cons of FNDLOAD verses iSETUP as well as other automated options to streamline this process.

APPENDIX A

```
# $Header: affrmcus.lct 115.8 2007/01/22 20:38:23 dbowles noship $
# affrmcus - FND Forms Customizations Loader Configuration
#
# Entities -
#
# FND_FORM_CUSTOM_RULES
# FND_FORM_CUSTOM_ACTIONS
# FND_FORM_CUSTOM_PARAMS
# FND_FORM_CUSTOM_SCOPES
# Note: to change the language you are downloading, setenv NLS_LANG before
# running the loader.
#
COMMENT = "dbdrv: exec fnd bin FNDLOAD bin &phase=daa+70
checkfile:~PROD:~PATH:~FILE &ui_apps 0 Y UPLOAD
@FND:patch/115/import/affrmcus.lct @~PROD:~PATH/~FILE"
# Parameters -
# (1) FUNCTION_NAME=VALUE (WHILE DOWNLOAD ONLY)
# (2) FORM_NAME=VALUE (WHILE DOWNLOAD ONLY)
#
# download example. Note if no parameters are passed, all rows downloaded
# FNDLOAD apps/apps 0 Y DOWNLOAD affrmcus.lct my.ldt FND_FORM_CUSTOM_RULES
function_name=FND_FNDSCAUS
# upload example
# FNDLOAD apps/apps 0 Y UPLOAD affrmcus.lct my.ldt
# Note: Any existing data for a function being uploaded is deleted before
# uploading occurs. No data
# is updated.
```

Look in comments
for entities

Look in comments
for parameters

```
DEFINE FND_FORM_CUSTOM_RULES
KEY ID VARCHAR2(50)
BASE FUNCTION_NAME VARCHAR2(30)
BASE DESCRIPTION VARCHAR2(255)
BASE TRIGGER_EVENT VARCHAR2(30)
BASE TRIGGER_OBJECT VARCHAR2(100)
BASE CONDITION VARCHAR2(2000)
BASE SEQUENCE VARCHAR2(50)
BASE CREATED_BY VARCHAR2(50)
BASE CREATION_DATE VARCHAR2(11)
BASE OWNER VARCHAR2(4000)
BASE LAST_UPDATE_DATE VARCHAR2(11)
BASE LAST_UPDATE_LOGIN VARCHAR2(50)
BASE ENABLED VARCHAR2(1)
BASE FIRE_IN_ENTER_QUERY VARCHAR2(1)
BASE RULE_KEY VARCHAR2(30)
BASE FORM_NAME VARCHAR2(30)
BASE RULE_TYPE VARCHAR2(1)
```

Also search for
"DEFINE" to find
Entities

```
DEFINE FND_FORM_CUSTOM_ACTIONS
KEY ACTION_ID VARCHAR2(50)
BASE SEQUENCE VARCHAR2(50)
TRANS PROPERTY_VALUE VARCHAR2(4000)
BASE ARGUMENT_TYPE VARCHAR2(1)
BASE CREATED_BY VARCHAR2(50)
BASE CREATION_DATE VARCHAR2(11)
```

```

BASE OWNER VARCHA2 (4000)
BASE LAST_UPDATE_DATE VARCHA2 (11)
BASE LAST_UPDATE_LOGIN VARCHA2 (50)
BASE TARGET_OBJECT VARCHA2 (100)
BASE ACTION_TYPE VARCHA2 (1)
BASE ENABLED VARCHA2 (1)
BASE OBJECT_TYPE VARCHA2 (30)
BASE FOLDER_PROMPT_BLOCK VARCHA2 (30)
BASE MESSAGE_TYPE VARCHA2 (1)
TRANS MESSAGE_TEXT VARCHA2 (4000)
BASE SUMMARY VARCHA2 (255)
BASE BUILTIN_TYPE VARCHA2 (1)
BASE BUILTIN_ARGUMENTS VARCHA2 (4000)
BASE LANGUAGE VARCHA2 (4)
KEY RULE_ID VARCHA2 (50)
BASE PROPERTY_NAME VARCHA2 (50)
BASE MENU_ENTRY VARCHA2 (30)
TRANS MENU_LABEL VARCHA2 (80)
BASE MENU_SEPERATOR VARCHA2 (1)
BASE MENU_ENABLED_IN VARCHA2 (255)
BASE MENU_ACTION VARCHA2 (1)
BASE MENU_ARGUMENT_LONG VARCHA2 (2000)
BASE MENU_ARGUMENT_SHORT VARCHA2 (255)
BASE REQUEST_APPLICATION_ID VARCHA2 (50)

```

DEFINE FND_FORM_CUSTOM_PARAMS

```

KEY ACTION_ID VARCHA2 (50)
KEY NAME VARCHA2 (240)
BASE VALUE VARCHA2 (4000)
BASE SEQUENCE VARCHA2 (50)
BASE DEFAULT_TYPE VARCHA2 (1)
BASE INHERIT VARCHA2 (1)
BASE LAST_UPDATE_DATE VARCHA2 (11)
BASE OWNER VARCHA2 (4000)
BASE CREATED_BY VARCHA2 (50)
BASE CREATION_DATE VARCHA2 (11)
BASE LAST_UPDATE_LOGIN VARCHA2 (50)
END FND_FORM_CUSTOM_PARAMS

```

END FND_FORM_CUSTOM_ACTIONS

DEFINE FND_FORM_CUSTOM_SCOPES

```

KEY RULE_ID VARCHA2 (50)
KEY LEVEL_ID VARCHA2 (50)
KEY LEVEL_VALUE VARCHA2 (4000)
BASE LEVEL_VALUE_APPLICATION_ID VARCHA2 (50)
BASE LAST_UPDATE_DATE VARCHA2 (11)
BASE OWNER VARCHA2 (4000)
BASE CREATION_DATE VARCHA2 (11)
BASE CREATED_BY VARCHA2 (50)
BASE LAST_UPDATE_LOGIN VARCHA2 (50)
END FND_FORM_CUSTOM_SCOPES

```

END FND_FORM_CUSTOM_RULES

DOWNLOAD FND_FORM_CUSTOM_RULES

```
"select ID,
        FUNCTION_NAME,
        DESCRIPTION,
        TRIGGER_EVENT,
        TRIGGER_OBJECT,
        CONDITION,
        SEQUENCE,
        CREATED_BY,
        to_char(CREATION_DATE, 'YYYY/MM/DD') CREATION_DATE,
        fnd_load_util.owner_name(LAST_UPDATED_BY) OWNER,
        to_char(LAST_UPDATE_DATE, 'YYYY/MM/DD') LAST_UPDATE_DATE,
        LAST_UPDATE_LOGIN,
        ENABLED,
        FIRE_IN_ENTER_QUERY,
        RULE_KEY,
        FORM_NAME,
        RULE_TYPE
from    fnd_form_custom_rules
where   CREATED_BY <> 1
        and (:FUNCTION_NAME is null or (:FUNCTION_NAME is not null and
function_name like :FUNCTION_NAME))
        and (:FORM_NAME is null or (:FORM_NAME is not null and form_name
like :FORM_NAME))
        order by FORM_NAME, FUNCTION_NAME"
```

Search for "DOWNLOAD" to view sql for downloaded entities

Search for bind variables in the first download section to determine attributes available

DOWNLOAD FND_FORM_CUSTOM_ACTIONS

```
"select ACTION_ID ACTIONS_ID,
        SEQUENCE,
        PROPERTY_VALUE,
        ARGUMENT_TYPE,
        CREATED_BY,
        to_char(CREATION_DATE, 'YYYY/MM/DD') CREATION_DATE,
        fnd_load_util.owner_name(LAST_UPDATED_BY) OWNER,
        to_char(LAST_UPDATE_DATE, 'YYYY/MM/DD') LAST_UPDATE_DATE,
        LAST_UPDATE_LOGIN,
        TARGET_OBJECT,
        ACTION_TYPE,
        ENABLED,
        OBJECT_TYPE,
        FOLDER_PROMPT_BLOCK,
        MESSAGE_TYPE,
        MESSAGE_TEXT,
        SUMMARY,
        BUILTIN_TYPE,
        BUILTIN_ARGUMENTS,
        LANGUAGE,
        RULE_ID,
        PROPERTY_NAME,
        MENU_ENTRY,
        MENU_LABEL,
        MENU_SEPERATOR,
        MENU_ENABLED_IN,
        MENU_ACTION,
```

```

        MENU_ARGUMENT_LONG,
        MENU_ARGUMENT_SHORT,
        REQUEST_APPLICATION_ID
    from fnd_form_custom_actions actions
    where actions.rule_id = :ID"

```

DOWNLOAD FND_FORM_CUSTOM_PARAMS

```

"select ACTION_ID,
       NAME,
       VALUE,
       SEQUENCE,
       DEFAULT_TYPE,
       INHERIT,
       to_char(LAST_UPDATE_DATE, 'YYYY/MM/DD') LAST_UPDATE_DATE,
       fnd_load_util.owner_name(LAST_UPDATED_BY) OWNER,
       CREATED_BY,
       to_char(CREATION_DATE, 'YYYY/MM/DD') CREATION_DATE,
       LAST_UPDATE_LOGIN
    from fnd_form_custom_params
    where action_id = :ACTIONS_ID"

```

DOWNLOAD FND_FORM_CUSTOM_SCOPES

```

"select rule_id,
       level_id,
       decode(level_id,
              10, to_char(level_value),
              20, to_char(level_value),
              30, (select responsibility_key
                   from fnd_responsibility
                   where responsibility_id = LEVEL_VALUE
                   and application_id =
LEVEL_VALUE_APPLICATION_ID),
              40, fnd_load_util.owner_name(LEVEL_VALUE))
       LEVEL_VALUE,
       level_value_application_id,
       to_char(LAST_UPDATE_DATE, 'YYYY/MM/DD') LAST_UPDATE_DATE,
       fnd_load_util.owner_name(LAST_UPDATED_BY) OWNER,
       to_char(CREATION_DATE, 'YYYY/MM/DD') CREATION_DATE,
       created_by,
       last_update_login
    from fnd_form_custom_scopes
    where rule_id = :ID"

```

Search for "UPLOAD" to view sql for uploaded entities

```
UPLOAD FND_FORM_CUSTOM_RULES
BEGIN
  "DECLARE
    f_luby          number; -- entity owner in file
    f_lupdate       date;  -- entity update date in file
    f_creator       number;
    --db_luby       number; -- entity owner in db
    --db_lupdate    date;  -- entity update date in db
    f_form_name     varchar2(30);

  BEGIN
    -- we really should not ever run in NLS mode, as the lct file is not used
to deliver Oracle NLS translated ldt files
    if :UPLOAD_MODE='NLS' then
      return;
    else
      -- Translate owner to file_last_updated_by
      f_luby := fnd_load_util.owner_id(:OWNER);
      -- Translate char last_update_date to date
      f_lupdate := nvl(to_date(:LAST_UPDATE_DATE, 'YYYY/MM/DD'), sysdate);
      -- Translate creator to f_creator
      f_creator := fnd_load_util.owner_id(:CREATED_BY);

      if :RULE_TYPE is NULL then
        -- Since rule id will not be consistent between systems,
        -- delete any existing records for the function_name
        -- before uploading
        FND_FORM_CUSTOM_RULES_PKG.DELETE_ROWS(:FUNCTION_NAME);
        -- we must be processing an ldt file that was created with a previous
lct version
        -- the form name will need to be extracted
        BEGIN
          select g.form_name INTO f_form_name
          from fnd_form_functions f, fnd_form g
          where f.form_id = g.form_id
          AND f.function_name = :FUNCTION_NAME;
        EXCEPTION
          WHEN OTHERS THEN
            f_form_name := :FUNCTION_NAME;
        END;

        insert into FND_FORM_CUSTOM_RULES(
          ID,
          FUNCTION_NAME,
          DESCRIPTION,
          TRIGGER_EVENT,
          TRIGGER_OBJECT,
          CONDITION,
          SEQUENCE,
          CREATED_BY,
          CREATION_DATE,
          LAST_UPDATED_BY,
          LAST_UPDATE_DATE,
          LAST_UPDATE_LOGIN,
          ENABLED,
          FIRE_IN_ENTER_QUERY,
```

```

RULE_KEY,
FORM_NAME,
RULE_TYPE)
values (
  FND_FORM_CUSTOM_RULES_S.NextVal,
  :FUNCTION_NAME,
  :DESCRIPTION,
  :TRIGGER_EVENT,
  :TRIGGER_OBJECT,
  :CONDITION,
  :SEQUENCE,
  f_luby,
  f_ludate,
  f_luby,
  f_ludate,
  0,
  :ENABLED,
  :FIRE_IN_ENTER_QUERY,
  NULL,
  f_form_name,
  'A');
else
  if :RULE_TYPE = 'A' then
    FND_FORM_CUSTOM_RULES_PKG.DELETE_ROWS (:FUNCTION_NAME);
  elsif :RULE_TYPE = 'F' then
    FND_FORM_CUSTOM_RULES_PKG.DELETE_FORM_ROWS (:FORM_NAME);
  end if;
insert into FND_FORM_CUSTOM_RULES (
  ID,
  FUNCTION_NAME,
  DESCRIPTION,
  TRIGGER_EVENT,
  TRIGGER_OBJECT,
  CONDITION,
  SEQUENCE,
  CREATED_BY,
  CREATION_DATE,
  LAST_UPDATED_BY,
  LAST_UPDATE_DATE,
  LAST_UPDATE_LOGIN,
  ENABLED,
  FIRE_IN_ENTER_QUERY,
  RULE_KEY,
  FORM_NAME,
  RULE_TYPE)
values (
  FND_FORM_CUSTOM_RULES_S.NextVal,
  :FUNCTION_NAME,
  :DESCRIPTION,
  :TRIGGER_EVENT,
  :TRIGGER_OBJECT,
  :CONDITION,
  :SEQUENCE,
  f_luby,
  f_ludate,
  f_luby,
  f_ludate,

```

```

    0,
    :ENABLED,
    :FIRE_IN_ENTER_QUERY,
    :RULE_KEY,
    :FORM_NAME,
    :RULE_TYPE);
end if;
end if;
END; "

```

UPLOAD FND_FORM_CUSTOM_ACTIONS

BEGIN

"DECLARE

```

    f_luby          number; -- entity owner in file
    f_lupdate       date;   -- entity update date in file
    f_creator       number;
    --db_luby       number; -- entity owner in db
    --db_lupdate    date;   -- entity update date in db

```

BEGIN

-- we really should not ever run in NLS mode, as the lct file in not used
to deliver Oracle NLS translated ldt files

if :UPLOAD_MODE='NLS' then

return;

else

-- Translate owner to file_last_updated_by

f_luby := fnd_load_util.owner_id(:OWNER);

-- Translate char last_update_date to date

f_lupdate := nvl(to_date(:LAST_UPDATE_DATE, 'YYYY/MM/DD'), sysdate);

-- Translate creator to f_creator

f_creator := fnd_load_util.owner_id(:CREATED_BY);

insert into FND_FORM_CUSTOM_ACTIONS(

```

    SEQUENCE,
    PROPERTY_VALUE,
    ARGUMENT_TYPE,
    CREATED_BY,
    CREATION_DATE,
    LAST_UPDATED_BY,
    LAST_UPDATE_DATE,
    LAST_UPDATE_LOGIN,
    TARGET_OBJECT,
    ACTION_TYPE,
    ENABLED,
    OBJECT_TYPE,
    FOLDER_PROMPT_BLOCK,
    MESSAGE_TYPE,
    MESSAGE_TEXT,
    SUMMARY,
    BUILTIN_TYPE,
    BUILTIN_ARGUMENTS,
    LANGUAGE,
    RULE_ID,
    PROPERTY_NAME,
    MENU_ENTRY,
    MENU_LABEL,

```

```

        MENU_SEPERATOR,
        MENU_ENABLED_IN,
        MENU_ACTION,
        MENU_ARGUMENT_LONG,
        MENU_ARGUMENT_SHORT,
        ACTION_ID,
        REQUEST_APPLICATION_ID)
values (
:SEQUENCE,
:PROPERTY_VALUE,
:ARGUMENT_TYPE,
f_luby,
f_ludate,
f_luby,
f_ludate,
0,
:TARGET_OBJECT,
:ACTION_TYPE,
:ENABLED,
:OBJECT_TYPE,
:FOLDER_PROMPT_BLOCK,
:MESSAGE_TYPE,
:MESSAGE_TEXT,
:SUMMARY,
:BUILTIN_TYPE,
:BUILTIN_ARGUMENTS,
:LANGUAGE,
FND_FORM_CUSTOM_RULES_S.CurrVal,
:PROPERTY_NAME,
:MENU_ENTRY,
:MENU_LABEL,
:MENU_SEPERATOR,
:MENU_ENABLED_IN,
:MENU_ACTION,
:MENU_ARGUMENT_LONG,
:MENU_ARGUMENT_SHORT,
FND_FORM_CUSTOM_ACTIONS_S.NextVal,
:REQUEST_APPLICATION_ID);

end if;
END;"

```

UPLOAD FND_FORM_CUSTOM_PARAMS

```

"DECLARE
  f_luby          number; -- entity owner in file
  f_ludate        date;   -- entity update date in file
  f_creator       number;
  --db_luby       number; -- entity owner in db
  --db_ludate     date;   -- entity update date in db

BEGIN

  -- we really should not ever run in NLS mode, as the lct file in not used
  to deliver Oracle NLS translated ldt files
  if :UPLOAD_MODE='NLS' then

```

```

return;
else
-- Translate owner to file_last_updated_by
f_luby := fnd_load_util.owner_id(:OWNER);
-- Translate char last_update_date to date
f_ludate := nvl(to_date(:LAST_UPDATE_DATE, 'YYYY/MM/DD'), sysdate);
-- Translate creator to f_creator
f_creator := fnd_load_util.owner_id(:CREATED_BY);
insert into FND_FORM_CUSTOM_PARAMS(
ACTION_ID,
NAME,
VALUE,
SEQUENCE,
DEFAULT_TYPE,
INHERIT,
LAST_UPDATED_BY,
LAST_UPDATE_DATE,
CREATED_BY,
CREATION_DATE,
LAST_UPDATE_LOGIN)
values(
FND_FORM_CUSTOM_ACTIONS_S.CurrVal,
:NAME,
:VALUE,
:SEQUENCE,
:DEFAULT_TYPE,
:INHERIT,
f_luby,
f_ludate,
f_luby,
f_ludate,
0);
end if;
END;"

```

UPLOAD FND_FORM_CUSTOM_SCOPES

```

"DECLARE
f_luby          number; -- entity owner in file
f_ludate        date;   -- entity update date in file
f_creator       number;
v_level_value   number;
--db_luby       number; -- entity owner in db
--db_ludate     date;   -- entity update date in db
BEGIN

-- we really should not ever run in NLS mode, as the lct file in not used
to deliver Oracle NLS translated ldt files
if :UPLOAD_MODE='NLS' then
return;
else
-- Translate owner to file_last_updated_by
f_luby := fnd_load_util.owner_id(:LAST_UPDATED_BY);
-- Translate char last_update_date to date
f_ludate := nvl(to_date(:LAST_UPDATE_DATE, 'YYYY/MM/DD'), sysdate);
-- Translate creator to f_creator
f_creator := fnd_load_util.owner_id(:CREATED_BY);

```

```

if :LEVEL_ID = 30 then
  begin
    select responsibility_id INTO v_level_value
    from fnd_responsibility
    where responsibility_key = :LEVEL_VALUE
    and application_id = :LEVEL_VALUE_APPLICATION_ID;
  exception
    when NO_DATA_FOUND then
      v_level_value := NULL;
    end;
  elsif :LEVEL_ID = 40 then
    v_level_value := fnd_load_util.owner_id(:LEVEL_VALUE);
  else
    v_level_value := :LEVEL_VALUE;
  end if;
insert into FND_FORM_CUSTOM_SCOPES(
  RULE_ID,
  LEVEL_ID,
  LEVEL_VALUE,
  LEVEL_VALUE_APPLICATION_ID,
  LAST_UPDATED_BY,
  LAST_UPDATE_DATE,
  CREATED_BY,
  CREATION_DATE,
  LAST_UPDATE_LOGIN)
values(
  FND_FORM_CUSTOM_RULES_S.CurrVal,
  :LEVEL_ID,
  v_level_value,
  :LEVEL_VALUE_APPLICATION_ID,
  f_luby,
  f_ludate,
  f_luby,
  f_ludate,
  0);
end if;
END;"

```