Oracle® GoldenGate for Flat File

Siebel Remote Integration Guide

11*g* Release 1 (11.1.1)

E18888-01

December 2010

Oracle GoldenGate for Flat File Oracle Siebel Remote Integration Guide 11*g* Release 1 (11.1.1)

E18888-01

Contents

CHAPTER 1

Introduction

This guide covers installing, configuring and running Oracle GoldenGate for Flat File to integrate Oracle GoldenGate for Siebel Remote.

Oracle GoldenGate solutions

Oracle GoldenGate

The core Oracle GoldenGate product:

- Captures transactional changes from a source database
- Sends and queues these changes as a set of database-independent files called the Oracle GoldenGate trail
- Optionally alters the source data using mapping parameters and functions
- Applies the transactions in the trail to a target system database

Oracle GoldenGate performs this capture and apply in near real-time across heterogeneous databases, platforms, and operating systems.

Refer to the *Oracle GoldenGate Administrator's Guide* and the *Oracle GoldenGate Reference Guide* for more information on this product.

Oracle GoldenGate for Flat File

Oracle GoldenGate for Flat File outputs transactional data captured by Oracle GoldenGate to rolling flat files to be consumed by a third party product. Oracle GoldenGate for Flat File is implemented as a user exit provided as a shared library (.so or .dll) that integrates into the Oracle GoldenGate Extract process.

The user exit supports two modes of output:

- DSV – Delimiter Separated Values (commas are an example)
- LDV – Length Delimited Values

It can output data:

- All to one file
- One file per table
- One file per operation code

The user exit can roll over based on time and/or size criteria. It flushes files and maintains

checkpoints whenever Oracle GoldenGate checkpoints to ensure recovery. It writes a control file containing a list of rolled over files for synchronization with the supported data integration product and can also produce a summary file for use in auditing. Additional properties control formatting (delimiters, other values), directories, file extensions, metadata columns (such as table name, file position, etc.) and data options.

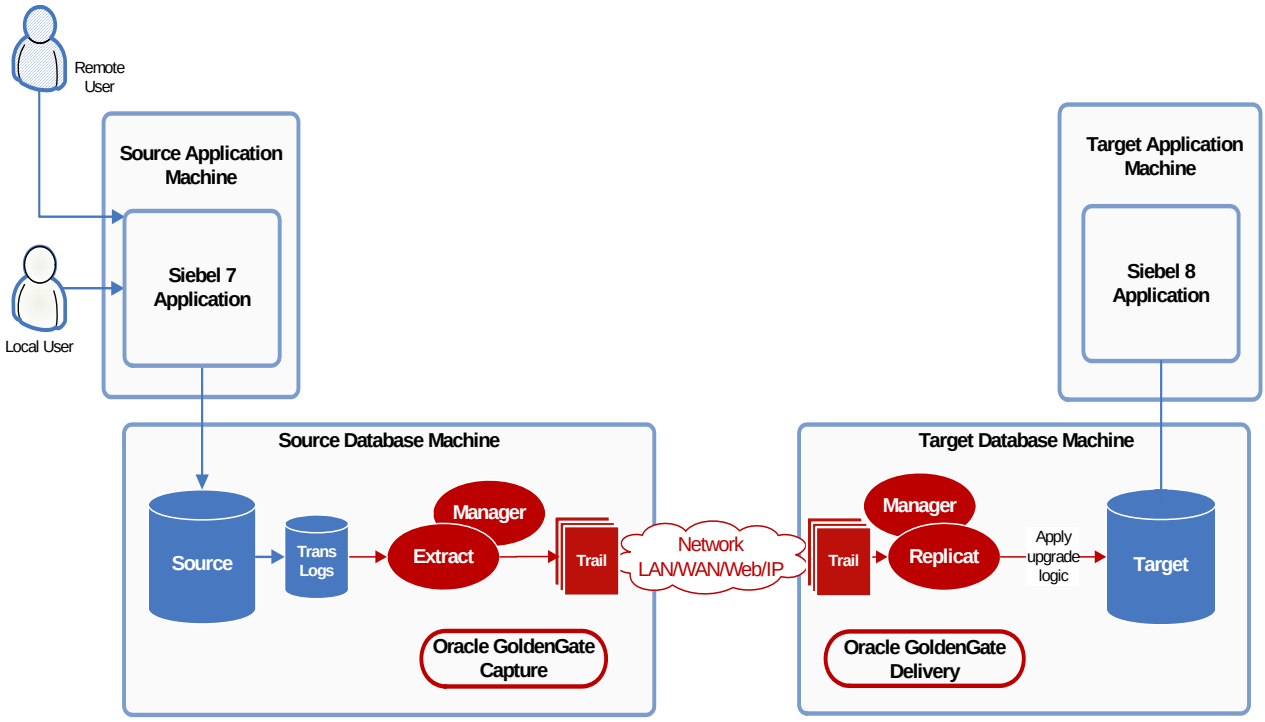
Refer to the *Oracle GoldenGate for Flat File Administrator's Guide* for more information on this product.

Oracle GoldenGate solutions for Siebel Remote

Oracle GoldenGate for Siebel Remote is used with zero downtime upgrade and territory alignment solutions:

- Involving Siebel Remote clients
- For either upgrades or highly available solutions
- Using a unidirectional configuration as shown in figure 1

Figure 1 Siebel zero downtime unidirectional upgrade configuration

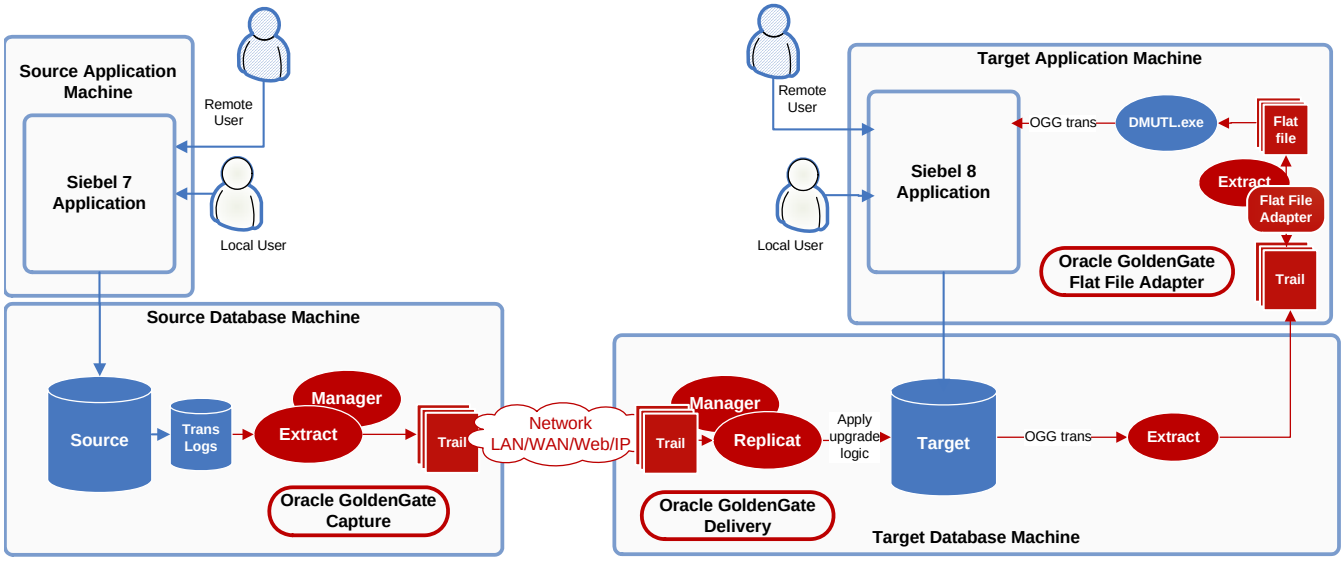


Oracle GoldenGate for Flat File is needed for Siebel Remote because Oracle GoldenGate interacts with the Siebel database directly. The Oracle GoldenGate changes are not processed by the Siebel object manager and not logged to the Siebel transaction log `S_DOCK_TXN`.

Because only logged transactions are available to update Siebel Remote users, Oracle GoldenGate for Flat Files must be added to the configuration to:

- Capture the changes the Oracle GoldenGate Replicat upgrade processing made directly to the target database
- Send the transactions to the Siebel Application server where the Oracle GoldenGate Flat File adapter creates records in the format and with the information needed for Siebel Remote processing
- Store the records in a series of flat files that are picked up by the DMUTL utility and imported to the Siebel Application to be processed to the Siebel Transaction log

Figure 2 Oracle GoldenGate for Flat File integration for Siebel Remote



Oracle GoldenGate for Flat File integration for Siebel Remote is implemented as a user exit provided as a shared library (.so or .dll). that integrates into the Oracle GoldenGate Extract process. By default the user exit produces a comma delimited file (.csv) that contains all transactions captured by the primary Extract.

Groups of transactions are written to the comma delimited file by default, but the user exit:

- Rolls over based on criteria set for time or size or both
- Flushes files
- Maintains checkpoints whenever Oracle GoldenGate checkpoints to ensure recovery

As files are rolled over, the Siebel utility, DMUTL, is executed. DMUTL imports the transactions into the Siebel controlled transaction log (`S_DOCK_TXN`).

Using DMUTL ensures that all changes, including those applied by the upgrade logic, are logged and routed to Siebel remote clients enabling zero downtime solutions for both connected and remote clients.

CHAPTER 2

Installing and Configuring

Prerequisites

The following steps should be taken before installing Oracle GoldenGate for Flat files for Siebel Remote integration.

- Obtain a licensed copy of Oracle GoldenGate release 11.1.1 or later.
- Install Oracle GoldenGate following the system prerequisites and directions in the Oracle GoldenGate Installation Guide.
- Fully test the installation before integrating the adapter.
- Enable supplemental logging for the following Siebel system columns. This is required for DMUTL processing.

  ```
  ROW_ID
  LAST_UPD
  LAST_UPD_BY
  MODIFICATION_NUM
  CONFLICT_ID
  ```

- Ensure that the installing user has file permissions to write data to the output directories.

Installing for Siebel Remote integration

Download the Oracle GoldenGate for Flat File integration with Siebel Remote software prebuilt for your particular platform and operating system.

Oracle GoldenGate for Flat File integration with Siebel Remote is shipped as a zip file that can be extracted using WinZip or a comparable program.

- To install on a Windows system unzip the file to the installation directory.
- To install on a UNIX system, first unzip the files on a Windows system and then FTP the `tar` file to the UNIX system. Use the gunzip command `tar -xvf` to install the files to the Oracle GoldenGate installation directory.

The file contains:

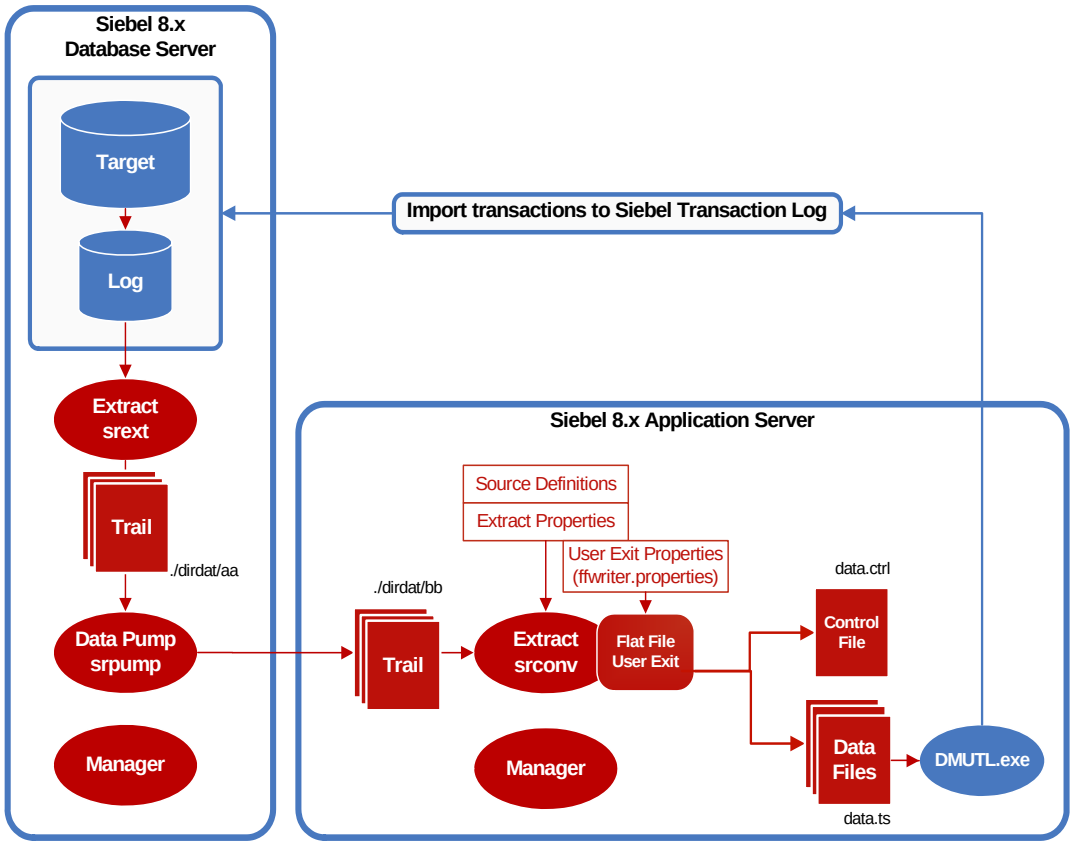
- Shared library: `flatfilewriter.dll` on Windows, `flatfilewriter.so` on UNIX
- Sample user exit properties file: `srconv.properties`
- Sample Extract parameter file: `srconv.prm`

Please refer to the "Processing the flat file using DMUTL" section of this document for details on automating the import of transaction files using DMUTL.

Configuring the Oracle GoldenGate processes

Figure 3 shows a typical Oracle GoldenGate for Flat File integration with Siebel Remote configuration.

Figure 3 Typical configuration for Oracle GoldenGate for Flat File integration with Siebel Remote



Adding the Extracts

The following example steps add the processes and trails.

1. Add the source Siebel 8.x database Extract and its trail.

```
GGSCI > ADD EXTRACT srext, TRANLOG, BEGIN NOW
GGSCI > ADD EXTTRAIL ./dirdat/aa, EXTRACT srext, MEGABYTES 20
```

2. Add the source Siebel 8.x database pump and trail.

```
GGSCI > ADD EXTRACT srpump, EXTTRAILSOURCE ./dirdat/aa
GGSCI > ADD RMTTRAIL ./dirdat/bb, EXTRACT srpump, MEGABYTES 20
```

3. Add the data integration system.

```
GGSCI > ADD EXTRACT srconv, EXTTRAILSOURCE ./dirdat/bb
```

The sample process names and trail names used above can be replaced with any valid name. Process names must be 8 characters or less; trail names must be two characters.

Processing the flat file using DMUTL

The following command is an example of using DMUTL to process the Oracle GoldenGate flat file on Windows. Note that the file switch (/F) should have a fully qualified path to the file.

```
dmutl /u SADMIN /p SADMIN /C siebel_DSN /D inst25ggs /G Y /K N /A N  /F
C:\path\srconv_00000_2009-xxxx_data.csv
```

Before running DMUTL be sure to set Siebel environmental variables by running `siebenv.bat` (`siebenv.sh` or `siebenv.csh` on UNIX), which is located in the *`Siebel_install_root`*`\bin` directory.

Below are all the switches available for DMUTL.

```
/U Username                  (Required)
/P Password                  (Required)
/C ODBC Data Source          (Default Env Variable: SIEBEL_DATA_SOURCE)
/D Siebel Table Owner        (Default Env Variable: SIEBEL_TABLE_OWNER)
/N Node Name                 (Required if Server Mode is Y)
/R User Txn Log file to read
/S Run in Server Mode        (Default: Y)
/G HA: Run in Import Mode    (Default: N)
/F HA: Import Filename
/K HA: Print Only
/A HA: Print CSV Only
/O Run in Optimistic Mode     (Default: Y)
/L Log Transactions           (Default: Y)
/V Update Dock Status Values  (Default: Y)
/T Transactions Per Commit
/I Source Node Id
/E Client Data Source         (Default: local)
```

To review the detailed transactions generated by DMUTL before applying them into the database, set the `/K` switch to `Y` as shown in the next example. This prints out to the screen only.

```
dmutl /u SADMIN /p SADMIN /C siebel_DSN /D inst25ggs /G Y /K Y /A N  /F
C:\path\srconv_00000_2009-xxxx_data.csv
```

NOTE Processing of .csv file files is not automatic . For each implementation a process must be developed to process files as they are produced. See the “Processing Siebel Remote transaction files” section of this document.

Example parameter files

Example Oracle GoldenGate Extract (SREXT) parameter file

The primary SREXT Extract resides on the Siebel database server. It captures changes made by Oracle GoldenGate and omits changes made by the Siebel application. It excludes certain named Siebel tables. The changes are stored locally in the aa trail.

```
EXTRACT SREXT
SETENV (ORACLE_SID = "SIEB")
USERID SIEBEL, PASSWORD SIEBEL
TRANLOGOPTIONS  EXCLUDEUSER ggs_user
GETREPLICATES
IGNOREAPPLOPS
GETUPDATEBEFORES
EXTTRAIL ./dirdat/aa
TABLEEXCLUDE SIEBEL.table_name;
TABLEEXCLUDE SIEBEL.table_name;
.
.
.
TABLE SIEBEL.S_*, keycols (ROW_ID);
```

Example Oracle GoldenGate data pump Extract (SRPUMP) parameter file

The SRPUMP Extract example picks up the transactions from the local aa trail and transfers them across the network to the bb trail on the Siebel 8 application server *host*.

```
EXTRACT SRPUMP
PASSTHRU
PASSTHRUMESSAGES
RMTHOST host MGRPORT port
CHECKPOINTSECS 1
WILDCARDRESOLVE DYNAMIC
DYNAMICRESOLUTION
RMTTRAIL ./dirdat/bb
TTABLE SIEBEL.*;
```

Example Oracle GoldenGate Extract (SRCONV) parameter file

The SRCONV Extract includes the user exit to the Oracle GoldenGate for Flat File adapter for integration to Siebel Remote.

```
EXTRACT SRCONV
SOURCEDEFS ./dirdef/Siebel_db_source_definition_filename.def
CUSEREXIT ./FlatFileUserExit.dll CUSEREXIT PASSTHRU,&
INCLUDEUPDATEBEFORES, PARAMS "srconv.properties"
GETUPDATEBEFORES
TABLE SIEBEL.*;
```

Example properties files

Templates

Oracle GoldenGate for Flat File includes a template of the standard properties commonly used for integration with Siebel Remote. To use this template, set the flat file writer template property (*writer*.template) as explained in the *Oracle GoldenGate for Flat File Administrator's Guide*.

For each of the properties in the template, the system first checks to see if that property is set in the properties file. If the user hasn't specified that property, the template setting is used. To see the properties included in the Siebel Remote template refer to the *Oracle GoldenGate for Flat File Administrator's Guide*.

Example Oracle GoldenGate Flat File properties (srconv.properties)

NOTE The names of some of the properties in the example may be different than the property name listed in the *Oracle GoldenGate for Flat File Administrator's Guide*. For example, goldengate.log.logname is log.logname. Either name is acceptable.

```
#------------------------
#LOGGING OPTIONS
#------------------------
# log file prefix
goldengate.log.logname=srconv

# global login level
goldengate.log.level=INFO

# output params, to stdout and/or to file
goldengate.log.tostdout=false
goldengate.log.tofile=true

# override the level and output params on a per module basis for debugging
goldengate.log.modules=LOGMALLOC
goldengate.log.level.LOGMALLOC=ERROR

# output a transaction at a time to handle begin / end tx markers
goldengate.userexit.outputmode=txs
# cache transaction before outputting in order to get correct op count
goldengate.userexit.buffertxs=true

# ensure all datetime datatypes are of format yyyy-mm-dd hh:mm:ss,
# not gg default yyyy-mm-dd:hh:mm:ss

goldengate.userexit.datetime.removecolon=true
# use utc, not local timestamp value
goldengate.userexit.timestamp=utc
```

```
#------------------------
#FLAT FILE WRITER OPTIONS
#------------------------

# specify the writers to use, enter names with properties below separated
# by commas, no spaces
goldengate.flatfilewriter.writers=srconv

#------------------------
# Named writer: srconv options
#------------------------
# output mode, DSV (delimiter separated) or BINARY (for length delimited
# format)
srconv.mode=DSV

# true/false flag to output characters binary (used to support Unicode
# data)
srconv.rawchars=false

# true/false flag to determine whether to output update before images
srconv.includebefores=true

# true/false flag to determine whether to output update column names
srconv.includecolnames=true

# true/false flag to determine whether to output column values
srconv.omitvalues=false

# true/false flag to determine whether to output only columns with
# differences between before and after images (deletes and inserts will
# have all available columns)
srconv.diffsonly=false

# true/false flag to determine whether to skip record delimiters if
# columns are missing
srconv.omitplaceholders=true

# true/false flag to determine whether to output all data to one file,
# or to create one file per table
srconv.files.onepertable=false

# naming convention for files
srconv.files.formatstring=srconv_%05n_%d

# specify the root directory for outputting data files
srconv.files.data.rootdir=./out

# determine the extension for data files when rolled over
srconv.files.data.ext=_data.csv
```

```
# determine the extension for data files before rolling over
srconv.files.data.tmpext=_data.csv.temp

# number of seconds before rolling over
srconv.files.data.rollover.time=10

# max file size in KB before rolling over
srconv.files.data.rollover.size=10000

# additional timeout to roll over in case no records for a period of time
srconv.files.data.norecords.timeout=10

# specify the standard UTF-8 bom as the first bytes in a data file
srconv.files.data.bom.code=efbbbf

# specify whether to output a control file
srconv.files.control.use=true

# specify the extension to use for control files
srconv.files.control.ext=_data.control

# and the directory in which to place control files, if not specified
# defaults to data directory
srconv.files.control.rootdir=./out

# The next indicator, delimiter and quote values can take .char or .code
# values
# If .char, enter the ascii (including C style escape - e.g. \t) values
# If .code enter the hex value for the characters to use

# define the characters to use for NULL values in delimiter separated
# files and (optionally) characters to use as escape characters for
# this value if found in data values
srconv.dsv.nullindicator.chars=NULL
#srconv.dsv.nullindicator.escaped.chars=

# define the characters to use for field delimiters in delimiter separated
# files and (optionally) characters to use as escape characters for
# this value if found in data values
srconv.dsv.fielddelim.chars=,
#srconv.dsv.fielddelim.escaped.chars=

# define the characters to use for line delimiters in delimiter separated
# files and (optionally) characters to use as escape characters for
# this value if found in data values
srconv.dsv.linedelim.chars=\n
#srconv.dsv.linedelim.escaped.chars=
```

```
# define the characters to use for quotes in delimiter separated files
# files and (optionally) characters to use as escape characters for
# this value if found in data values
srconv.dsv.quotes.chars="
srconv.dsv.quotes.escaped.chars=""

srconv.dsv.quotealways=true

# define the extra columns that may appear at the beginning of each record
# can include:
#  position - A unique position indicator of records in a trail
#  opcode - I, U or D for Insert, Update and Delete records
#  txind - Kind of record in a transaction (0-begin, 1-middle, 2-end,
#          3-whole)
#  txoppos - Position of record in a transaction, starting from 0
#  schema - The schema (owner) name of the changed record
#  table - The table name of the changed record
#  schemaandtable - Both the schema and table name concatenated as
#                   schema.table
#  timestamp - The commit timestamp of the record
#  "value" - hard coded output value
#  @TOKEN - value of a token
#  $GETENV.VALUE - value of a GoldenGate @GETENV col function
#  %COL_NAME - value of a column - this column will no longer be output
#              with the other columns

# group the columns names, before values and after values together,
# instead of interlacing them
srconv.groupcols=true

# output after values before before values
srconv.afterfirst=true

# define the metadata columns output for a begin transaction
srconv.begintx.metacols="B","S",position,"GGMC",%LAST_UPD_BY,"1",numops

# define the metadata columns output for each record
srconv.metacols="R",opcode,%ROW_ID,%LAST_UPD_BY,%LAST_UPD,%MODIFICATION
_NUM,%CONFLICT_ID,position,txoppos,table,"","","","","","",%DB_LAST_UPD
,%DB_LAST_UPD_SRC,numcols

# define the metadata columns output for an end transaction
srconv.endtx.metacols="E"

# don't output the DB_LAST_UPD or DB_LAST_UPD_SRC values
srconv.metacols.DB_LAST_UPD.omit=true
srconv.metacols.DB_LAST_UPD_SRC.omit=true
```

```
# output PK updates as if they are updates - don't use specific character
# for the op type
srconv.metacols.opcode.updatepk.chars=U

# output the trail position as decimal
srconv.metacols.position.format=dec

# roll over all files when the process is shutdown cleanly
srconv.files.rolloveronshutdown=true

# write statistics to the report file
srconv.statistics.toreportfile=true

# write to the report file on rollover
srconv.statistics.period=ONROLLOVER

# output totals in statistics
srconv.statistics.overall=true
```

CHAPTER 3

Processing

Processing Siebel Remote transaction files

To take advantage of the integration capabilities with Siebel Remote, you should automate the process of importing captured transaction using DMUTL. Automation tools such as UNIX CRON or other scheduling products could be used in conjunction with a script or `.bat` file that performs the following:

1. Wait on the control file
2. Read list of files to process from the control file
3. Rename the control file
4. Iterate over the comma-delimited list of files read from the control file
5. Process each data file, deleting the data file when complete
6. Delete the renamed control file

On startup, the process should check for the renamed control file to see if it must recover from previously failed processing

When the control file is renamed, the user exit will write a new one on the first file rollover. This will contain the list of files for the next batch.

If the user exit has been configured to also output a summary file, the data integration tool can optionally also read that summary file and cross-check the number of operations it has processed with the data in the summary file for each processed data file.

Example UNIX shell script

This routine is designed to continuously import files produced by the Oracle GoldenGate flat file writer for integration with Siebel Remote. Transactions are imported using the Siebel supplied utility, DMUTIL.

NOTE **Important!** This script is an example of how files can be processed using a UNIX shell script. It is not recommended for any purpose other than as an example. Oracle will not provide support, enhancements or bug fixes.

Example script: `srconv.sh [start | stop | status]`

```
#!/bin/bash
################################################################################
# File: srconv.sh
# Input Values: start/stop/status
# start - begins processing until the stop signal is encountered.
# stop -  sends stop signal to main process to shutdown gracefully.
# status - displays current status of the process. Includes last ten lines
#         of the event log.
#
# Assumptions: Process must be run as the goldengate user.
#
# Purpose: This routine is designed to continuously import files produced
# by the GoldenGate flat file writer for integration with Siebel remote.
# Transactions are imported using the Siebel supplied utility, dmutil.
#
################################################################################
#
#===============================================================================
# Variables
#===============================================================================

# Siebel home directory
export SIEBEL_HOME=/u01/siebel/product

#Siebel admin account. Used by DMUTL to import transactions into
# s_dock_txn
export SIEBEL_USER='sadmin'

# Siebel admin account password
export SIEBEL_PWD='sadmin'

# Siebel Repository Name
export SIEBEL_REPOSITORY='Siebel Repository'

# Siebel Connection String
export SIEBEL_DATA_SOURCE='SIEB_DSN'

# Siebel Table Owner
export SIEBEL_TABLE_OWNER='siebel'

# Directory which stores the csv files generated by the flat file writer
export SRCONV_DATA=$HOME/out

# Directory of archived csv files
export SRCONV_ARCH=$SRCONV_DATA/arch

# The name of the flat file writer control file
export SRCONV_CTL=$SRCONV_DATA/output_data.control
```

```
# The name of the temp flat file writer control file
export SRCONV_INPROC=$SRCONV_DATA/output_data.inprocess

# Controls the number of flat file processors to one
export SRCONV_PROC_CTL=/tmp/.srconv_proc.control

# Event log for flat file processing
export SRCONV_EVENT_LOG=$SRCONV_DATA/srconv_event.log

# Log directory for individual log files
export SRCONV_LOG_DIR=$SRCONV_DATA/log

# Constants
c_zero=0
c_one=1

#===============================================================================
# FUNCTIONS
#===============================================================================

fnWriteLog() {
##############################
# Name  : fnWriteLog
# Descr : Writes formatted entries to event log
##############################

v_date=`date`

echo $v_date""$1 >> $SRCONV_EVENT_LOG

} # end fnWriteLog

fnProcFile()
{
##############################
# Name  : fnProcFile
# Descr : Imports transactions via dmutl for found in files listed in
# control file.
##############################

# Name of the file listing the file files to process
f_CntrlFile=$1

# Count the number of files to be processed in the file
f_FileCnt=`cat $f_CntrlFile | sed 'y/,/ /' | wc -w`
f_Loop=1
```

```
# for every file in the control file, process it
while [[ $f_FileCnt -ge $f_Loop ]]
do

# Check for stop signal.
# If a stop signal is found, the inprocess files will be left intact
# and will be processed once the process is restarted.
fnCheckSig

# Get next file name to process
f_ShortFileNm=`cat $f_CntrlFile | cut -d',' -f$f_Loop | awk -F"/" '{print
$3}' `

# Set archive file name
f_ArchFileNm=$SRCONV_ARCH/$f_ShortFileNm".arch"

# Set log file.  One log file for each file
f_FileNm_log=$SRCONV_LOG_DIR/$f_ShortFileNm".log"

# Full path the file name
f_FileNm=$SRCONV_DATA/$f_ShortFileNm

if [ -f $f_FileNm ]
then
fnWriteLog "Processsing: $f_FileNm"
dmutl /u $SIEBEL_USER /p $SIEBEL_PWD  /G Y /K N /A N /F $f_FileNm  >
$f_FileNm_log 2>&1
# Check return code of DMUTL

if [ $? -ne $c_zero ]
then
       fnWriteLog "ERROR:  DMUTL unsuccessfully imported $f_FileNm."
       fnAbort
fi

# Move processed file & log to archive directory
f_date=`date`
fnWriteLog "Archiving $f_ShortFileNm "
mv $f_FileNm $f_ArchFileNm

else
# File name in control file does not correspond to any phsyical file.
fnWriteLog "File Not Found. Skipping $f_FileNm."
fi

f_Loop=`expr $f_Loop + 1`

done

} # end fnProcFile
```

```
fnStart() {
##############################
# Name  : fnStart
# Descr : Establishes process control file for start-up.
##############################

# Determine if there is a flat file processor is running
# Only one flat file processor is to run at any time
if [ -f $SRCONV_PROC_CTL ]; then

# confirm that the pid is running
v_old_pid=`cat $SRCONV_PROC_CTL | awk -F":" '{print $1}'`

v_pid_cnt=`ps -ef | grep -c $v_old_pid`

  if [ $v_pid_cnt -ge $c_one ]; then
# Process is already running, exit.
echo "Restart attempted. Process already running"
fnWriteLog "Restart attempted. Process already running"
exit 0
else
# Process is not running. Re-initialize process control file
echo "Process Restarted."
fnWriteLog "Process Restarted."
echo $$":Running" > $SRCONV_PROC_CTL
fi

else
# Initialize the process control file
echo "Starting up."
fnWriteLog "Starting up."
echo $$":Running" > $SRCONV_PROC_CTL

fi

} # end fnStart

fnStop() {
##############################
# Name  : fnStop
# Descr : Sends stop signal to gracefully shut down.
##############################

# Determine if there is a flat file processor is running
if [ -f $SRCONV_PROC_CTL ]; then
```

```
# confirm that the pid is running
               v_old_pid=`cat $SRCONV_PROC_CTL | awk -F":" '{print $1}'`

                v_pid_cnt=`ps -ef | grep -c $v_old_pid`

                if [ $v_pid_cnt -ge $c_one ]; then
# Process is already running, send stop signal
echo "Shutdown signal sent."
fnWriteLog "Shutdown signal sent."
echo $v_old_pid":stop" > $SRCONV_PROC_CTL
                else
# Process is not running.
echo "Shutdown attempted. Process was not running."
fnWriteLog "Shutdown attempted. Process was not running."
rm $SRCONV_PROC_CTL
                fi
        else
# No process control file found.  Process must not be running
echo "ERROR: Process control file not found."
fnWriteLog "ERROR: Process control file not found."
        fi

exit 0

} # end fnStop

fnAbort() {
##############################
# Name  : fnAbort
# Descr : Cleans up control files during error conditions
##############################

fnWriteLog "Process Aborted."

rm $SRCONV_PROC_CTL > /dev/null 2>&1
exit 1

} # end fnAbort

fnCheckSig() {
##############################
# Name  : fnCheckSig
# Descr : Checks for stop signal in process control file.
#  If stop signal is found, process exists gracefully
##############################

# Check for stop signal
if [ -f $SRCONV_PROC_CTL ]
then
                v_sig=`cat $SRCONV_PROC_CTL | grep -c "stop"`
```

```
if [ $v_sig -ge $c_one ]
then
fnWriteLog "Shutdown signal received. Shutting down."
rm $SRCONV_PROC_CTL > /dev/null 2>&1
exit 0
fi

else
# Control file lost during processing.  Abort further work.
fnWriteLog "ERROR:  Process control file not found.  Aborting"
fnAbort
fi

} # end fnCheckSig

fnStat() {
##############################
# Name  : fnStat
# Descr : Reports on current status of process to standard out
##############################

d_date=`date`

echo "****************************************"
echo "* SRCONV.SH Status Report"
echo "* $d_date"
echo "****************************************"
echo "**"

if [ -f $SRCONV_PROC_CTL ]
then

        v_pid=`cat $SRCONV_PROC_CTL | awk -F":" '{print $1}'`

        v_pid_cnt=`ps -ef | grep -c $v_pid`

        if [ $v_pid_cnt -ge $c_one ]; then

echo "Current Running PID is " $v_pid
else
echo "Process is Currently NOT Running"
fi
else
echo "Process is Currently NOT Running"
fi
echo "**"
echo "Latest Log Entries"
echo "**"
```

```
tail $SRCONV_EVENT_LOG

echo "**"

} # end fnStat

#===========================================================================
# MAIN
#===========================================================================

# Check the number command line arguments
if [ $# -ne  $c_one  ]; then
        echo "srconv.sh - you must specify a parameter"
        echo "Usage: srconv.sh [start, stop, status]"
        exit
fi

# Check command line arguments
case $1 in
        "start")  fnStart
                  ;;
        "stop")   fnStop
                  ;;
        "status") fnStat;;
        *) echo "Valid arguments are start,stop,or status"; exit;;
esac

# Confirm Data and Archive Directories Exist

# Data Direcotry Check
if [ ! -d $SRCONV_DATA ]
then
echo "ERROR:  Data Directory $SRCONV_DATA does not exist"
fnWriteLog "ERROR:  Data Directory $SRCONV_DATA does not exist"
fnAbort
fi

# Archive Directory Check
if [ ! -d $SRCONV_ARCH ]
then
echo "ERROR: Archive Directory $SRCONV_ARCH does not exist"
fnWriteLog "ERROR: Archive Directory $SRCONV_ARCH does not exist"
fnAbort
fi

# Archive Directory Check
if [ ! -d $SRCONV_LOG_DIR ]
then
```

```
echo "ERROR: Log Directory $SRCONV_LOG_DIR does not exist"
fnWriteLog "ERROR: Log Directory $SRCONV_LOG_DIR does not exist"
fnAbort
fi

# Note:  Add additional directory checks as needed.

# Check for any prior failed inprocess files
if [ -f $SRCONV_INPROC ]
then
# A control file from a prior run still exists.
# Process any files that remain.
fnWriteLog "Partially Processed Control File Found."
fnWriteLog  "Processing Files in $SRCONV_INPROC"

fnProcFile $SRCONV_INPROC

fnWriteLog  "Completed Processing Files in $SRCONV_INPROC"

fi

while [ $c_one -eq $c_one ]
do
# Check for stop signals
fnCheckSig

# If a control file exists, process the file listed in it
if [ -f $SRCONV_CTL ]
then
# Move the control file off for processing.  A new one will
# automatically be created by the flat file write

fnWriteLog "Processing Files in $SRCONV_CTL"

# Move off the control file to a temporary processing file
mv $SRCONV_CTL $SRCONV_INPROC

# Process the files listed in the inprocess control file
fnProcFile $SRCONV_INPROC

# Remove the temporary processing file
rm $SRCONV_INPROC > /dev/null 2>&1
else
# control file not found.   Sleep
sleep 5

fi

done
## END
```

Index

P

R

S

T

U

W