#!/usr/local/bin/perl -w

# my name is Ryan. this is for testing

################################################################################

# PROGRAM: $Id: updateDLAP.pl,v 1.12 2008/02/20 14:59:19 antoniop Exp $

# @(#) $Revision: 1.12 $

#

# COPYRIGHT ALCANET INTERNATIONAL 1995-2002

# Alcanet International - Services Engineering

#

# DESCRIPTION

# Update a DLAP

# 1) Get by scp a tar file which contains the new db files (iPlanet format)

# 2) Unzip, untar them in a temporary directory

# 3) Change owner (chown to the user running the ns-directory)

# 3) Stop safekit i.e Safekit stops iPlanet directory (process launch by Safekit itself) and stops the load balancing

# 4) Stop iPlanet (if not already done by safekit)

# 5) Copy the db files in the correct place ( update the directory )

# 6) Start safekit i.e Safekit tries to start the directory (process launch by safekit itself) and restarts the load balancing

# 7) Start iPlanet (if not already done by safekit)

# 8) save a copy of the last running db files

#

# USAGE: updateDLAP.pl

#

# INPUT

# See configuration file

#

# OUTPUT

# 0 if successfull

# 1 if error

#

# NOTES

# this script must be run as root because of safekit command (start, stop)

# Following alarm are defined:

# DLAP\_CANNOT\_CREATE\_FILE

# DLAP\_CANNOT\_READ\_FILE

# DLAP\_CP\_FAILED

# DLAP\_CHOWN\_FAILED

# DLAP\_NO\_NEW\_DATABASE\_AVAILABLE

# DLAP\_REMOTE\_SAFEKIT\_STOPPED

# DLAP\_SAFEKIT\_SNMP\_STATE\_ERROR

# DLAP\_SAFEKIT\_START\_FAILED

# DLAP\_SAFEKIT\_STOP\_FAILED

# DLAP\_SCP\_FAILED

# DLAP\_SLAPD\_START\_FAILED

# DLAP\_SLAPD\_STOP\_FAILED

# DLAP\_SNMP\_FAILED

# DLAP\_UNTAR\_FAILED

# DLAP\_UNZIP\_FAILED

# DLAP\_ZIP\_FAILED

# Following keys has to be defined in the configuration file:

# NSDIR\_USER: user that run NS directory

# NSDIR\_GROUP: group of the user that run NS directory

# DIRECTORY\_PATH: path for NS diretcory (eg /iplanet/server/slapd-test)

# NBRETRY: Number of retry

# INTERVALL: Number of secong between 2 tries

# LOG: log dir

# ARCHIVE: archive directory

# GZIP: full path for gzip (eg /usr/local/bin/gzip)

# TAR: full path for tar (eg: /usr/local/bin/tar)

# encrypt\_key: key used to encrypt the file

# LOCK\_PAUSE: sleep time between each loop for wiating slapd to stop.

# LOCK\_COUNT: Number of loops to wait for slapd to stop

#

# Following keys are optionals

# PLUGGW\_USE: 1 if switch of netperm-table is required. Default is 0

# if PLUGGW\_USE == 1, then following parameters are mandatories:

# PLUGGW\_LOCAL\_TEMPLATE

# PLUGGW\_BACKUP\_TEMPLATE

# PLUGGW\_NETPERM\_TABLE

#

# IS\_SAFEKIT\_USE: 1 if safekit is used as fail over,unless 0. Default is 1

# if IS\_SAFEKIT\_USE == 1, then following parameters are mandatories:

# SAFEKIT\_PATH: path for Safekit (eg. /opt/safekit)

# LIGHT\_DLAP: 1 if there is 1 DLAP server (no load balancer). Default is 0

# REMOTE\_SNMP\_SAFEKIT\_SERVER: remote safekit

# REMOTE\_SNMP\_SAFEKIT\_PORT: snmp port used by safekit: (eg: 3600)

#

# LIGHT\_HUB: 1 if Hub is on the same server than the DLAP. Default is 0

# if LIGHT\_HUB == 0, then following parameters are mandatories:

# SCP: full path for scp (eg: /usr/sbin/scp)

# SSH: full path for ssh (eg: /usr/sbin/scp)

# REMOTE\_USER: user on the remote system for db files

# REMOTE\_HOST: host for db files

# REMOTE\_FILE: name of the db files

#

#

#@BEGIN

# HISTORIC

# $Log: updateDLAP.pl,v $

# Revision 1.12 2008/02/20 14:59:19 antoniop

# BUG:SAFEKIT replacement USR:AP MSG:Ctrl remote host. Flag when DLAP stopped to update.

#

# Revision 1.11 2005/11/04 10:40:50 herrou1

# BUG:add ssh path in configuration file. Do not exit if slapd stops after end of stopSafekit. USR:LH MSG:

#

# Revision 1.10 2005/09/13 11:25:36 herrou1

# BUG:DLAP USR:LH MSG:encrypt\_key is mandatory only if encryption is used

#

# Revision 1.9 2005/09/13 09:41:29 herrou1

# BUG:DLAP USR:LH MSG:add optional parameter to handle encryption

#

# Revision 1.8 2004/09/06 09:06:13 herrou1

# BUG:merge with RM changes USR:LH MSG:add lock count and pause to the config file to operate the wait loop for the iPlanet slapd stop; add an info if stopSlapd needs not to act for there is no slapd process ; dd a loop to wait for the stop of the iPlanet slapd; dd switchNetpermTable function

#

# Revision 1.7 2003/10/22 09:36:32 pilot800

# BUG:DLAP17 USR:LH MSG:add file encryption (decryption)

#

# Revision 1.6 2002/12/02 08:23:06 pilot800

# BUG:LHG USR:DLAP06 MSG:misspelling on some alarm (SAFKIT replaced by SAFEKIT)

#

# Revision 1.5 2002/11/19 10:12:05 pilot800

# BUG:DLAP05 USR:LH MSG:modify script for getting remote state of Safekit. Do an achive file

#

# Revision 1.4 2002/11/14 14:11:07 pilot800

# BUG:DLAP03 USR:LH MSG:remove temporrily the check on remote Safekit state because of new Safekit version

#

# Revision 1.3 2002/10/11 11:54:57 herrou1

# BUG:DLAP02 USR:LH MSG:add missing "use lib"

#

# Revision 1.2 2002/07/22 11:36:54 herrou1

# BUG:DLAP02 USR:LH MSG:modify some alarm message and remove some unneeded functions

#

# Revision 1.1.1.1 2002/07/19 14:40:07 herrou1

# Fault tolerant design: step 3 - DLAP

#

#@END

################################################################################

use Env qw(HOME);

use strict;

use lib "/COMMON/lib";

use lib "$HOME/COMMON/lib";

use Trace;

use Conf;

use File::Basename;

################################################################################

#

# INITIALIZE

#

################################################################################

my $conf=new Conf("DLAP");

#

# Log

#

my $FILTERTRACE="TRACE";

#$FILTERTRACE="DEBUG";

my $TRACELEVEL="USEFUL";

my $log = $conf->getParam("LOG");

my $trace = Trace->new($log,$FILTERTRACE,$TRACELEVEL);

#

# Checks parameters

#

if (! $conf->checkKeyList(["NSDIR\_USER", "NSDIR\_GROUP", "TAR", "NBRETRY", "INTERVALL",

"DIRECTORY\_PATH", "GZIP", "ARCHIVE", "LOCK\_PAUSE", "LOCK\_COUNT" ], $trace) ) {

die "Some mandatory parameters are missing. Please check log file $log\n";

}

my $nsdir\_user = $conf->getParam("NSDIR\_USER");

my $nsdir\_group = $conf->getParam("NSDIR\_GROUP");

my $tar = $conf->getParam("TAR");

my $nbretry = $conf->getParam("NBRETRY");

my $intervall = $conf->getParam("INTERVALL");

my $directory\_path = $conf->getParam("DIRECTORY\_PATH");

my $gzip = $conf->getParam("GZIP");

my $archive = $conf->getParam("ARCHIVE");

#AP 1.12 - file created when DLAP is voluntary stopped.

my $SEMA = "$directory\_path/failoverstoppedflag";

my @psef\_grep\_lines=""; # rm need it for the grep result

my $lock\_pause = $conf->getParam("LOCK\_PAUSE"); # rm lock the script for xx seconds

my $lock\_count = $conf->getParam("LOCK\_COUNT"); # rm y times xx seconds before we give up

#

# Encrypt parameter - Optional Parameters

#

my $ENCRYPT = 0;

my $ENCRYPT\_KEY;

$ENCRYPT = $conf->getParam("encrypt") if ( $conf->checkKeyList(["encrypt"], $trace) ) ;

if ($ENCRYPT) {

if (! $conf->checkKeyList(["encrypt\_key"], $trace) ) {

die "Some mandatory parameters are missing. Please check log file $log\n";

}

$ENCRYPT\_KEY = $conf->getParam('encrypt\_key');

use Crypt::CBC;

}

#

# PLUG-GW Parameters - Optional Parameters

#

my $pluggw\_use = 0;

#AP 1.12 - delay before stop the DLAP - value loaded from conf below.

my $pluggw\_delay = 0;

my $pluggw\_local\_template;

my $pluggw\_backup\_template;

my $pluggw\_netperm\_table;

if ($conf->checkKeyList(["PLUGGW\_USE"], $trace) ) {

$pluggw\_use = $conf->getParam("PLUGGW\_USE");

}

if ( $pluggw\_use ) {

$pluggw\_delay = $conf->getParam("PLUGGW\_DELAY");

$pluggw\_local\_template = $conf->getParam("PLUGGW\_LOCAL\_TEMPLATE");

$pluggw\_backup\_template = $conf->getParam("PLUGGW\_BACKUP\_TEMPLATE");

$pluggw\_netperm\_table = $conf->getParam("PLUGGW\_NETPERM\_TABLE");

}

#

# SCP Parameters / Local Hub - Optional Parameters

#

my $IS\_LOCAL\_HUB = 0;

my $scp;

my $ssh;

my $remote\_user;

my $remote\_host;

my $remote\_file;

if ($conf->checkKeyList(["LIGHT\_HUB"], $trace) ) {

$IS\_LOCAL\_HUB = $conf->getParam("LIGHT\_HUB");

}

if ( ! $IS\_LOCAL\_HUB) {

if ($conf->checkKeyList(["SCP", "SSH", "REMOTE\_USER", "REMOTE\_HOST", "REMOTE\_FILE"], $trace) ) {

use Net::SNMP;

$ssh = $conf->getParam("SSH");

$scp = $conf->getParam("SCP");

$remote\_user = $conf->getParam("REMOTE\_USER");

$remote\_host = $conf->getParam("REMOTE\_HOST");

$remote\_file = $conf->getParam("REMOTE\_FILE");

} else {

die "Some mandatories parameters are missing (check SCP,SSH, REMOTE\_USER, REMOTE\_HOST, REMOTE\_FILE) in $log\n";

}

} else { # Hub is on the same server (local hub)

$scp = "cp";

$remote\_host = 'localhost';

$remote\_file = $conf->getParam("REMOTE\_FILE");

$remote\_user = "";

}

#AP 1.12 - To access remote server on HUB

my $testsnmp\_server = $conf->getParam("REMOTE\_SNMP\_SAFEKIT\_SERVER");

my $portsearch = $conf->getParam("DIRECTORY\_PORT");

#

# DLAP is alone (no load balancer / failover) - OptionalParameter

# SAFEKIT is used

#

my $IS\_SAFEKIT\_USED=1;

my $IS\_DLAP\_ALONE = 0;

my $safekit\_path;

my $snmp\_server;

my $snmp\_port;

if ($conf->checkKeyList(["SAFEKIT\_USED"], $trace) ) {

$IS\_SAFEKIT\_USED = $conf->getParam("SAFEKIT\_USED");

}

if ( $IS\_SAFEKIT\_USED ) {

if ($conf->checkKeyList(["SAFEKIT\_PATH"], $trace) ) {

$safekit\_path = $conf->getParam("SAFEKIT\_PATH");

} else {

die "Some mandatories parameters are missing (check SAFEKIT\_PATH) in $log\n";

}

if ($conf->checkKeyList(["LIGHT\_DLAP"], $trace) ) {

$IS\_DLAP\_ALONE = $conf->getParam("LIGHT\_DLAP");

}

if ( ! $IS\_DLAP\_ALONE) {

if ($conf->checkKeyList(["REMOTE\_SNMP\_SAFEKIT\_SERVER", "REMOTE\_SNMP\_SAFEKIT\_PORT"], $trace) ) {

$snmp\_server = $conf->getParam("REMOTE\_SNMP\_SAFEKIT\_SERVER");

$snmp\_port = $conf->getParam("REMOTE\_SNMP\_SAFEKIT\_PORT");

} else {

die "Some mandatories parameters are missing (check REMOTE\_SNMP\_SAFEKIT\_SERVER,REMOTE\_SNMP\_SAFEKIT\_PORT) in $log\n";

}

}

}

my $date;

my $LOCAL\_FILE;

$date=`date +%I`;

chomp $date;

$LOCAL\_FILE="$archive/alcatel.db.$date.tar.gz";

my $ARCHIVE\_FILE="$archive/alcatel.db.tar.gz";

# Keep trace of the lastest uplaod file

my $lastfile="$archive/.ls\_output";

$trace->print("START UPDATE DLAP PROCESS\n");

# check on the remote system if the database file has changed . Avoid to run an update for nothing...

my $local\_file;

if ($ENCRYPT) {

$local\_file="$LOCAL\_FILE.encrypted";

$remote\_file="$remote\_file.encrypted";

} else {

$local\_file="$LOCAL\_FILE";

}

if (&waitNewDatabase($remote\_file, $nbretry, $intervall, $archive, $remote\_user, $remote\_host, $IS\_LOCAL\_HUB, $ssh)) {

$trace->alarm(

error=>"DLAP\_NO\_NEW\_DATABASE\_AVAILABLE",

module=>"DLAP",

params=>[$remote\_host, $remote\_file],

msg=>"No new database available on $remote\_host for $remote\_file). Please check the export process",

);

$trace->print("EXIT UPDATE DLAP PROCESS on error");

&clean();

exit 1;

}

my $ls; # to remember what is the last database that has been uploaded

$ls=&getFile($remote\_file,$remote\_user,$remote\_host,$scp,$ssh,$local\_file, $IS\_LOCAL\_HUB);

if (! defined $ls ) {

$trace->print("EXIT UPDATE DLAP PROCESS on error");

unlink $local\_file;

exit 1;

}

if ($ENCRYPT) {

&\_decryptFile("$local\_file", $ENCRYPT\_KEY, "$LOCAL\_FILE");

}

# If safekit is stopped on the other side, we do nothing to avoid cut of service.

# This function will wait a while until Safekit is started

if ( $IS\_SAFEKIT\_USED and ! $IS\_DLAP\_ALONE and &waitForRemoteSafekit($snmp\_server,$snmp\_port,$nbretry,$intervall)) {

$trace->alarm(

error=>"DLAP\_REMOTE\_SAFEKIT\_STOPPED",

module=>"DLAP",

params=>[],

msg=>"The remote Safekit is stopped. Update has been stopped",

);

$trace->print("EXIT UPDATE DLAP PROCESS on error");

unlink $LOCAL\_FILE;

&clean();

exit 1;

}

#AP 1.12 - This function verify that remote DLAP is started.

if ( ! $IS\_SAFEKIT\_USED and &waitForRemoteDlap($testsnmp\_server,$nbretry,$intervall)) {

$trace->alarm(

error=>"REMOTE\_DLAP\_STOPPED",

module=>"DLAP",

params=>[],

msg=>"The remote DLAP is stopped. Update has been stopped",

);

$trace->print("EXIT UPDATE DLAP PROCESS on error");

unlink $LOCAL\_FILE;

&clean();

exit 1;

}

if (&\_unzip($gzip,$LOCAL\_FILE)) {

$trace->print("STOP DLAP UPDATE PROCESS\n");

unlink $LOCAL\_FILE;

exit 1;

}

# untar the files in a temporary directory

$LOCAL\_FILE=~s/\.gz$//;

if (&\_untar($LOCAL\_FILE,$archive,$tar)) {

$trace->print("STOP DLAP UPDATE PROCESS\n");

unlink $LOCAL\_FILE;

exit 1;

}

# As safekit is runnign as root, we must change the file to

# the nsserver user

if ( &changeOwner($nsdir\_user,$nsdir\_group,"$archive/db")) {

$trace->print("STOP DLAP UPDATE PROCESS\n");

unlink $LOCAL\_FILE;

exit 1;

}

# Stop Safekit

if ( $IS\_SAFEKIT\_USED and safekit\_stop($safekit\_path,$nbretry,$intervall)) {

$trace->print("STOP DLAP UPDATE PROCESS\n");

unlink $LOCAL\_FILE;

exit 1;

}

# switch the netperm from local to the backup

#

if ( $pluggw\_use )

{

&switchNetpermTable("$pluggw\_backup\_template", "$pluggw\_netperm\_table");

#AP 1.12 - delay before to stop the DLAP.

sleep($pluggw\_delay);

}

# Stop the directory (in case Safekit was already stopped)

if(&stopSlapd($directory\_path)) {

$trace->print("STOP DLAP UPDATE PROCESS\n");

unlink $LOCAL\_FILE;

#AP 1.12 - When script is stopped, the flag file is removed.

unlink $SEMA;

exit 1;

}

if (! &copyFile("$archive/db", "$directory\_path")) {

&copyFile("$archive/db", "$archive/.last.correct.db");

}

if ( $IS\_SAFEKIT\_USED ) {

&safekit\_start($safekit\_path,$nbretry,$intervall);

}

&startSlapd($directory\_path);

# switch the netperm backup to local

#

if ( $pluggw\_use )

{

&switchNetpermTable("$pluggw\_local\_template", "$pluggw\_netperm\_table");

}

rename ("$LOCAL\_FILE", $ARCHIVE\_FILE);

# memorize the last database that has been uploaded

if (open ("LS", ">$archive/.ls\_output")) {

print LS $ls;

close LS;

} else {

$trace->alarm(

error=>"DLAP\_CANNOT\_CREATE\_FILE",

module=>"DLAP",

params=>["$archive/.ls\_output"],

msg=>"Can't update file $archive/.ls\_output",

);

}

$trace->print("STOP DLAP UPDATE PROCESS - Success\n");

exit 0;

#-------------------------------------------------------

# Function : safekit\_stop

#

# Description : Stop Safekit and wait for it state to be

# "stop"

#

# Parameters : safekit path to safekit binary

# \* number of retry to wait the "Stop" state

# \* time between 2 retries

#

# Return: \* 1 in case of error

# \* 0 if success

#

#-------------------------------------------------------

sub safekit\_stop {

my $safekit\_path=shift;

my $retry=shift;

my $sleep=shift;

my $errfile="/tmp/safekit.$$";

$trace->pushInfo("safekit\_stop");

my $exit\_status=system("$safekit\_path/safekit stop >$errfile 2>&1");

if ($exit\_status == 256) {

$trace->print("already stopped\n", "DEBUG");

# Check here directory is stopped

} elsif ($exit\_status != 0) {

$trace->print("Failed", "DEBUG");

$trace->popInfo();

$trace->print(`cat $errfile`);

$trace->alarm(

error=>"DLAP\_SAFEKIT\_STOP\_FAILED",

module=>"DLAP",

params=>[],

msg=>"error when stopping safekit: $exit\_status",

);

unlink "$errfile";

$trace->popInfo();

return 1;

}

$trace->print("Command success\n","DEBUG");

$trace->popInfo();

unlink $errfile;

# wait a while, since safekit is stopped

$trace->pushInfo("Waiting for safekit to stop....", "DEBUG");

my $try=0;

while (&get\_safekit\_state($safekit\_path) != 0 and $try < $retry ) {

$trace->print("retry: $try", "DEBUG");

sleep($sleep);

$try++;

}

if ($try == $retry) {

$trace->print("Failed","DEBUG");

} else {

$trace->print("Success","DEBUG");

$try=0;

}

$trace->popInfo();

return $try;

}

#-------------------------------------------------------

# Function : safekit\_start

#

# Description : Start Safekit and wait for its state to be

# "prim" or "second" or "alone"

#

# Parameters : \* safekit path to safekit binary

# \* number of retry to wait the "started" state

# \* time between 2 retries

#

# Return: \* 1 in case of error

# \* 0 if success

#

#-------------------------------------------------------

sub safekit\_start {

my $safekit\_path=shift;

my $retry=shift;

my $sleep=shift;

my $errfile="/tmp/safekit.$$";

$trace->pushInfo("safekit\_start");

$trace->print("Safekit start...", "DEBUG");

#my $exit\_status=system("$safekit\_path/safekit start >$errfile 2>&1");

my $exit\_status=system("/opt/safekit/safekit start >$errfile 2>&1");

if ($exit\_status != 0) {

$trace->print("Failed", "DEBUG");

$trace->print(`cat $errfile`);

$trace->print("Error when starting Safekit");

$trace->alarm(

error=>"DLAP\_SAFEKIT\_START\_FAILED",

module=>"DLAP",

params=>[],

msg=>"error when starting safekit: $exit\_status ",

);

unlink $errfile;

$trace->popInfo();

return $exit\_status;

} else {

$trace->print("Success", "DEBUG");

}

unlink $errfile;

# wait a while, since safekit is started

my $try=0;

$trace->print("Waiting for safekit end its startup process....", "DEBUG");

while (&get\_safekit\_state($safekit\_path) < 2 and $try < $retry ) {

$trace->print("$try", "DEBUG");

sleep($sleep);

$try++;

}

if ($try == $retry) {

$trace->print("Failed","DEBUG");

} else {

$trace->print("Success","DEBUG");

$try=0;

}

$trace->popInfo();

return $try;

}

#-------------------------------------------------------

# Function : get\_safekit\_state

#

# Description : run the command safekit state

#

# Parameters : \* safekit path to safekit binary

#

# Return: \* 0: stop

# \* 1: wait

# \* 2: Start (Alone)

# \* 3: Start (Prim)

# \* 4: Start (Second)

#

#-------------------------------------------------------

sub get\_safekit\_state {

my $safekit\_path=shift;

$trace->pushInfo("get\_safekit\_state");

system("$safekit\_path/safekit state >/dev/null 2>&1");

my $exit\_value = $? >> 8;

$trace->popInfo();

return $exit\_value;

}

#-------------------------------------------------------

# Function : stopSlapd

#

# Description : stop a ns-slapd directory

#

# Parameters : \*instance path (full path)

# eg: /work/ids51/slapd-master1

#

# Return: \* 0 on success

# \* <> 0 on error

#-------------------------------------------------------

sub stopSlapd {

my $instanceDir=shift;

my $lock=0; # rm - init the loop count

if ( ! -e "$instanceDir/logs/pid") {

# the server is probably not running - test anyway the process slapd

@psef\_grep\_lines = `ps -ef | grep $instanceDir | grep -v grep`; # rm - pick up process data

# in case there is no process existing , we should not run a stop-slapd

if($#psef\_grep\_lines < 0)

{

$trace->print("no stop-slapd needed >$#psef\_grep\_lines<");

return 0 ;

}

}

#AP 1.12 - Creation of the flag file to avoid restart the DLAP.

unless (open(FLAGFAILOVER,">$SEMA")) {

$trace->alarm(

error=>"FILE\_OPEN",

params=>["FLAGFAILOVER","$!"],

module=>"EXPORT",

);

$trace->print(" \*\*\*\*\*\*\*\*\* STOP ERR \*\*\*\*\*\*\*\*");

die "Can't create $SEMA\n";

}

else {

print FLAGFAILOVER "The fail over is stopped : " . &common::chrono() . "";

chmod 0664, $SEMA;

}

close(FLAGFAILOVER);

$trace->pushInfo("Stop the directory $instanceDir ...");

my $errfile="/tmp/DLAP.$$";

my $exitstatus=system("$instanceDir/stop-slapd 2>$errfile");

if ( $exitstatus != 0 ) {

$trace->print("Failed");

$trace->popInfo();

$trace->print(`cat $errfile`);

@psef\_grep\_lines = `ps -ef | grep $instanceDir | grep -v grep`;

if ($#psef\_grep\_lines > -1 ) {

$trace->print("directory still alive @psef\_grep\_lines");

while ( $lock < $lock\_count )

{

$trace->print("sleep $lock\_pause - lock is $lock - max lock\_count is $lock\_count","DEBUG");

$lock++;

sleep $lock\_pause;

@psef\_grep\_lines = `ps -ef | grep $instanceDir | grep -v grep`;

if($#psef\_grep\_lines > -1)

{

$trace->print("directory is alive @psef\_grep\_lines");

}

else

{

$trace->print("directory is no longer alive $instanceDir");

#AP 1.13 - Update $lock to force loop exit.

$lock = $lock\_count;

$exitstatus = 0;

}

}

} else {

$trace->print("directory is no longer alive $instanceDir");

$exitstatus = 0;

}

}

if ($exitstatus != 0 ) {

$trace->alarm(

error=>"DLAP\_SLAPD\_STOP\_FAILED",

module=>"DLAP",

params=>[$instanceDir],

msg=>"error on command: $instanceDir/stop-slapd. See log file for details",

);

} else {

$trace->print("Success");

$trace->popInfo();

# traced a situation where after a stop still the pid file was existing

# in such a case no start happens for the start depends on a not existing pid file

if ( -e "$instanceDir/logs/pid")

{

$trace->print("forced deletion of $instanceDir/logs/pid");

unlink "$instanceDir/logs/pid";

}

}

unlink "$errfile";

return $exitstatus;

}

#-------------------------------------------------------

# Function : getFile

#

# Description : get DB file by scp or cp

#

# Parameters : \* remoteFile: name (full path) of teh file to get

# \* remoteUser: name of the user on the remote server

# \* remoteHost: name of the remote host

# \* scpBin: full path for scp or cp command

# \* localFile: full path for the local file

# \* isLocalHub: 1 if use cp, 0 if use scp

#

# Return: \* -1 on error

# \* `ls -l` on the file that has been uploaded

#-------------------------------------------------------

sub getFile {

my $remoteFile=shift;

my $remoteUser=shift;

my $remoteHost=shift;

my $scpBin=shift;

my $sshBin=shift;

my $localFile=shift;

my $isLocalHub=shift;

$trace->pushInfo("getFile");

my $errfile="/tmp/DLAP.$$";

$trace->print("Get dbFile from $remoteHost...", "DEBUG");

$trace->print("From Host: $remoteHost", "DEBUG");

$trace->print("Remote User:$remoteUser", "DEBUG");

$trace->print("RemoteFile:$remoteFile", "DEBUG");

$trace->print("Local file:$localFile", "DEBUG");

$trace->print("scp: $scpBin", "DEBUG");

my $exit\_status=0;

if ($isLocalHub) {

$exit\_status=system("$scpBin $remoteFile $localFile 2>$errfile 1>&2");

} else { # remote

$exit\_status=system("$scpBin -B $remoteUser\@$remoteHost:$remoteFile $localFile 2>$errfile 1>&2");

}

my $error=`cat $errfile`;

$trace->print($error, "DEBUG");

if ($exit\_status != 0 or $error) {

$trace->print("Failed");

$trace->print($error);

if ($isLocalHub) {

$trace->alarm(

error=>"DLAP\_SCP\_FAILED",

module=>"DLAP",

params=>[$remoteUser,$remoteHost,$remoteFile,$localFile],

msg=>"error when $scpBin $remoteFile $localFile. Run the command manually to have more details",

);

} else {

$trace->alarm(

error=>"DLAP\_SCP\_FAILED",

module=>"DLAP",

params=>[$remoteUser,$remoteHost,$remoteFile,$localFile],

msg=>"error when $scpBin $remoteUser\@$remoteHost:$remoteFile $localFile. Run the command manually to have more details",

);

}

unlink "$errfile";

$trace->print("STOP UPDATE DLAP PROCESS\n");

$trace->popInfo();

return undef;

}

unlink "$errfile";

if ( $isLocalHub ) {

$ls=qx/ls -l $remoteFile / ;

} else {

$ls=qx/$sshBin -l $remoteUser $remoteHost \"ls -l $remoteFile\" / ;

}

$trace->print("Success\n");

$trace->popInfo();

return $ls;

}

#-------------------------------------------------------

# Function : waitForRemoteSafekit

#

# Description : Wait the remote Safekit is started

#

# Parameters : \* safekitPath: full path for safekit (eg /opt/safekit);

# \* nbretry: number of retry before stop

# \* intervall: nb second before the next retry

#

# Return: \* 0 on success

# \* >1 on error

#

#-------------------------------------------------------

sub waitForRemoteSafekit {

my $server=shift;

my $port=shift;

my $nbretry=shift;

my $sleep=shift;

my $try=0;

$trace->pushInfo("Wait for the remote safekit starts...");

while (&get\_remote\_state($server, $port) < 2 and $try < $nbretry) {

$trace->print("try $try", "DEBUG");

sleep $sleep;

$try++;

}

if ($try == $nbretry) {

$trace->print("Failed");

$trace->popInfo();

} else {

$trace->print("Success");

$trace->popInfo();

$try=0;

}

return $try;

}

#-------------------------------------------------------

# Function : get\_remote\_state

#

# Description : get the state of Safekit on a remote system

#

# Parameters : \* host or IP of the remote server

# \* port for snmp

#

# Return: \* 0 if safekit is STOP

# \* 1 is Safekit is WAIT

# \* 2 if Safekit is UP

# \* -1 on error

#

#-------------------------------------------------------

sub get\_remote\_state {

my $server=shift;

my $snmp\_port=shift;

$trace->pushInfo("get\_remote\_state");

my ($session, $error) = Net::SNMP->session(

-hostname => "$server",

-community => 'public',

-port => "$snmp\_port",

-version => 'snmpv1'

);

if (!defined($session)) {

my $error = $session->error;

$trace->print("SNMP session failed. $error");

$trace->popInfo();

$trace->alarm(

error=>"DLAP\_SNMP\_FAILED",

module=>"DLAP",

params=>[$error],

msg=>"Error during snmp session.",

);

$trace->popInfo();

return -1;

}

my $status = '1.3.6.1.4.1.107.175.10.1.1.4.1';

my $result = $session->get\_request(

-varbindlist => [$status]

);

if (!defined($result)) {

my $error = $session->error;

$trace->print("Failed getting safekit state. $error");

$trace->popInfo();

$trace->alarm(

error=>"DLAP\_SAFEKIT\_SNMP\_STATE\_ERROR",

module=>"DLAP",

params=>[$error],

msg=>"Failed to get safekit state.",

);

$session->close;

$trace->popInfo();

return -1;

}

$trace->print("status for host " . $session->hostname . " is " . $result->{$status}, "DEBUG");

$session->close;

$trace->popInfo();

return $result->{$status};

}

#-------------------------------------------------------

# Function : \_unzip

#

# Description : unzip a file in the same directory as the zip file

#

# Parameters : \* gzip command full path

# \* full path of the file to unzip

#

# Return: \* 0 on success

# \* >1 on error

#

#-------------------------------------------------------

sub \_unzip {

my $gzip=shift;

my $file=shift;

$trace->pushInfo("\_unzip");

my $errfile="/tmp/DLAP.$$";

$trace->pushInfo("Unzip the dbfile $file...");

my $exit\_status=system("$gzip -S .gz.clear -d $file 2>$errfile");

if ($exit\_status != 0) {

$trace->print("Failed");

$trace->popInfo();

$trace->print(`cat $errfile`);

$trace->alarm(

error=>"DLAP\_UNZIP\_FAILED",

module=>"DLAP",

params=>[$file],

msg=>"Error during Unzip. See Log file for details.",

);

unlink "$errfile";

$trace->popInfo();

return 1;

}

$trace->print("Success", "DEBUG");

$trace->popInfo();

unlink "$errfile";

$trace->popInfo();

return 0;

}

#-------------------------------------------------------

# Function : \_untar

#

# Description : untar a file in the same directory as the tar file

#

# Parameters : \* full path of the file to unzip

# \* directory where untar file will be

# \* command tar full pathname

#

# Return: \* 0 on success

# \* 1 on error

#

#-------------------------------------------------------

sub \_untar {

my $file=shift;

my $dir=shift;

my $tar=shift;

$trace->pushInfo("\_untar");

chdir $dir;

my $errfile="/tmp/DLAP.$$";

my $exitstatus=system("$tar xf $file 2>$errfile");

if ($exitstatus != 0) {

$trace->print("Failed");

$trace->popInfo();

$trace->print(`cat $errfile`);

$trace->alarm(

error=>"DLAP\_UNTAR\_FAILED",

module=>"DLAP",

params=>[$file],

msg=>"Error during \_untar. See log file for details",

);

} else {

$trace->print("Success", "DEBUG");

}

unlink "$errfile";

$trace->popInfo();

return $exitstatus;

}

#-------------------------------------------------------

# Function : changeOwner

#

# Description : change owner of a directory (recursively)

#

# Parameters : \* new user

# \* new group

# \* directory to chown

#

# Return: \* 0 on success

# \* <>0 on error

#

#-------------------------------------------------------

sub changeOwner {

my $user=shift;

my $group=shift;

my $dir=shift;

$trace->pushInfo("changeOwner");

$trace->print("Change owner of $dir to $user:$group", "DEBUG");

my $errfile="/tmp/DLAP.log";

my $exitstatus=system("chown -R $user:$group $dir 2>$errfile 1>&2");

if ($exitstatus != 0) {

$trace->print("Failed");

$trace->popInfo();

$trace->print(`cat $errfile`);

$trace->alarm(

error=>"DLAP\_CHOWN\_FAILED",

module=>"DLAP",

params=>[$user,$group, $dir],

msg=>"Error during chown. See log file for details.",

);

} else {

$trace->print("Success");

}

unlink "$errfile";

$trace->popInfo();

return $exitstatus;

}

#-------------------------------------------------------

# Function : startSlapd

#

# Description : start the directory

#

# Parameters : \* instance name (full path)

#

# Return: \* 0 on success

# \* <>0 on error

#

#-------------------------------------------------------

sub startSlapd {

my $instanceDir=shift;

return if ( -e "$instanceDir/logs/pid" );

$trace->pushInfo("startSlapd");

my $errfile="/tmp/DLAP.$$";

$trace->print("Start the directory...", "DEBUG");

my $exitstatus=system("$instanceDir/start-slapd 2>$errfile 1>&2");

if ( $exitstatus != 0 ) {

$trace->print("Failed");

$trace->popInfo();

$trace->print(`cat $errfile`);

$trace->alarm(

error=>"DLAP\_SLAPD\_START\_FAILED",

module=>"DLAP",

params=>[$instanceDir],

msg=>"Error when starting the directory. See log file for details",

);

} else {

$trace->print("Success");

#AP 1.12 - When DLAP is started, the flag file is removed.

unlink $SEMA;

}

unlink "$errfile";

$trace->popInfo();

return $exitstatus;

}

#-------------------------------------------------------

# Function : copyFiles

#

# Description : Copy some files from a dir to another one

#

# Parameters : \* from dir

# \* destination dir

#

# Return: \* 0 on success

# \* <>0 on error

#

#-------------------------------------------------------

sub copyFile {

my $org=shift,

my $dest=shift;

$trace->pushInfo("copyFile");

my $errfile="/tmp/DLAP.$$";

$trace->print("Copy file from $org to $dest...", "DEBUG");

my $exitstatus=system("cp -pR $org $dest 2>$errfile 1>&2");

if ( $exitstatus != 0 ) {

$trace->print("Failed");

$trace->popInfo();

$trace->print(`cat $errfile`);

$trace->alarm(

error=>"DLAP\_CP\_FAILED",

module=>"DLAP",

params=>[$org,$dest],

msg=>"Error during copy of db files. See error log for details.",

);

} else {

$trace->print("Success");

}

unlink "$errfile";

$trace->popInfo();

return $exitstatus;

}

#-------------------------------------------------------

# Function : zip

#

# Description : zip a file in the same directory as the unzip file

#

# Parameters : \* gzip command full path

# \* full path of the file to zip

#

# Return: \* 0 on success

# \* >1 on error

#

#-------------------------------------------------------

sub zip {

my $gzip=shift;

my $file=shift;

$trace->pushInfo("zip");

my $errfile="/tmp/DLAP.$$";

$trace->print("zip the file $file...", "DEBUG");

my $exit\_status=system("$gzip $file 2>$errfile");

if ($exit\_status != 0) {

$trace->print("Failed");

$trace->popInfo();

$trace->print(`cat $errfile`);

$trace->alarm(

error=>"DLAP\_ZIP\_FAILED",

module=>"DLAP",

params=>[$LOCAL\_FILE,$gzip],

msg=>"Error during zip. See Log file for details.",

);

unlink "$errfile";

$trace->popInfo();

return 1;

}

$trace->print("Success", "DEBUG");

$trace->popInfo();

unlink "$errfile";

return 0;

}

#-------------------------------------------------------

# Function : clean

#

# Description : Remove temporary files before ending the program

#

# Parameters : none

#

# Return: nothing

#

#-------------------------------------------------------

sub clean {

$trace->pushInfo("clean");

unlink "/tmp/DLAP.$$";

unlink $LOCAL\_FILE;

unlink "$LOCAL\_FILE.gz";

$trace->popInfo();

}

#-------------------------------------------------------

# Function : waitNewDatabase

#

# Description : This function will check that a new database

# is available on the hub.

#

# Parameters : - file: name of the file to check

# - loop: number max of try to do

# - sleep: number of time (second) to wait between 2 tries

# - user: only in case of remote server. Remote user (for scp)

# - server: only in case of remote server. Remote host (for scp)

# - isLocal: 1 of cp ; 0 if scp

#

# Return: 0 if success

# 1 on error

#-------------------------------------------------------

sub waitNewDatabase {

my ($file, $loop, $sleep, $archive, $user, $host, $isLocal, $sshBin)=@\_;

$trace->pushInfo("waitNewDatabase");

my $check\_ls\_output;

my $checkold\_ls\_output;

my $i;

#

# first check weather this script does not already run

#

$trace->print("Is the script already running ?", "DEBUG");

my $process = basename($0);

# sh is removed in case the program is launched by crontab

my $nbLines = `ps -ef | grep $process | grep -v grep | grep -v sh | wc -l`;

if($nbLines > 1)

{

print "found $nbLines processes with the string $process as name component\n";

$trace->print("The script is already running", "DEBUG");

$trace->popInfo();

return 1;

}

$trace->print("The script is NOT already running", "DEBUG");

#

# conserve a first ls output

#

if (open ("LS", "$archive/.ls\_output")) {

$checkold\_ls\_output=<LS>;

chomp $checkold\_ls\_output;

close LS;

$trace->print("latest database is $checkold\_ls\_output", "DEBUG");

} else {

$trace->print("No latest database found in $archive/.ls\_output", "DEBUG");

$trace->popInfo();

return 0;

}

for ( $i= 1; $i <= $loop; $i +=1)

{

# print "LOOPCOUNT: $i\n";

#

# run a ls

if ($isLocal) {

$check\_ls\_output = qx/ls -l $file/ ;

} else { # remote

$check\_ls\_output = qx/$sshBin -l $user $host \"ls -l $file\"/ ;

}

$trace->print("New database is $check\_ls\_output", "DEBUG");

#

# compare against the old ls content , if changed we can start the copy

#

if ($check\_ls\_output =~ /^$checkold\_ls\_output$/)

{

$trace->print("old: $checkold\_ls\_output\nnew: $check\_ls\_output", "DEBUG");

sleep $sleep;

}

else

{

#

if ( $check\_ls\_output =~ /$file/)

{ # check if the ls contains good data if not try it again

#sleep 5 ; # better wait short that the copy on the remote site can finish

$trace->print("WOULD START THE SCP NOW", "DEBUG");

$trace->print("old: $checkold\_ls\_output\nnew: $check\_ls\_output", "DEBUG");

$trace->popInfo();

return 0;

}

sleep 5;

$trace->print("BAD CONTENT CASE\nold: $checkold\_ls\_output\nnew: $check\_ls\_output", "DEBUG");

}

}

$trace->popInfo();

return 1;

}

#-------------------------------------------------------

# Function : \_decryptFile

#

# Description : decrypte a file with twofish algorithm

#

# Parameters : - filename of the encrypted file

# - encyrpted key

# - filename of the decrypted file

#

# Return : \* 0 if success

# \* 1 if error

#

#-------------------------------------------------------

sub \_decryptFile() {

my $file=shift;

my $key=shift;

my $clearfile=shift;

$trace->pushInfo("\_decryptFile");

my $buffer;

my $cipher = new Crypt::CBC($key,'Twofish');

$cipher->start("decrypt");

unless (open(IN, "$file")) {

$trace->alarm(

error=>"DLAP\_CANNOT\_READ\_FILE",

module=>"DLAP",

params=>["$file"],

msg=>"Cannot open file $file for reading",

);

$trace->popInfo();

return 1;

}

unless (open(OUT, ">$clearfile")) {

$trace->alarm(

error=>"DLAP\_CANNOT\_CREATE\_FILE",

module=>"DLAP",

params=>["$clearfile"],

msg=>"Cannot create file $clearfile for reading",

);

close IN;

$trace->popInfo();

return 1;

}

while( read(IN, $buffer, 1024) ) {

print OUT $cipher->crypt($buffer);

}

print OUT $cipher->finish;

close IN;

close OUT;

unlink $file;

$trace->popInfo();

return 0;

}

#-------------------------------------------------------

# Function : switchNetpermTable

#

# Description : Copy some files from a dir to another one

#

# Parameters : \* from file

# \* destination file

#

# Return: \* 0 on success

# \* <>0 on error

#

#-------------------------------------------------------

sub switchNetpermTable {

my $org=shift,

my $dest=shift;

my $errfile="/tmp/DLAP.$$";

$trace->pushInfo("Copy netperm from $org to $dest...");

my $exitstatus=system("cp -f $org $dest 2>$errfile 1>&2");

if ( $exitstatus != 0 ) {

$trace->print("Failed");

$trace->popInfo();

$trace->print(`cat $errfile`);

$trace->alarm(

error=>"NETPERM\_CP\_FAILED",

module=>"DLAP",

params=>[$org,$dest],

msg=>"Error during copy of netperm files. See error log for details.",

);

} else {

$trace->print("success");

$trace->popInfo();

}

unlink "$errfile";

return $exitstatus;

}

#AP 1.12 - New functions.

#-------------------------------------------------------

# Function : waitForRemoteDlap

#

# Description : Wait the remote DLAP is started

#

# Parameters : \* snmp server: remote server

# \* nbretry: number of retry before stop

# \* intervall: nb second before the next retry

#

# Return: \* 0 on success

# \* >1 on error

#

#-------------------------------------------------------

sub waitForRemoteDlap {

my $server=shift;

# my $user=shift;

# my $dlappath=shift;

my $nbretry=shift;

my $sleep=shift;

# my $sshBin=shift;

my $try=0;

$trace->pushInfo("Wait for the remote DLAP starts...");

# while (&get\_remote\_dlap\_state($server,$user,$dlappath,$sshBin) and $try < $nbretry) {

while (&get\_remote\_dlap\_state($server) and $try < $nbretry) {

$trace->print("try $try", "DEBUG");

sleep $sleep;

$try++;

}

if ($try == $nbretry) {

$trace->print("Failed");

$trace->popInfo();

} else {

$trace->print("Success");

$trace->popInfo();

$try=0;

}

return $try;

}

#-------------------------------------------------------

# Function : get\_remote\_dlap\_state

#

# Description : get the state of a remote DLAP

#

# Parameters : \* snmp server: remote server

#

# Return: \* 1 if DLAP is DOWN

# \* 0 is DLAP is UP

#

#-------------------------------------------------------

sub get\_remote\_dlap\_state {

my $server = shift;

# my $user = shift;

# my $dlappath = shift;

# my $sshBin = shift;

my $ligpsresult;

# my $partdir = substr($dlappath, 0, length($dlappath) - 2);

# my $cmdstring = "$sshBin -l $user $server \"ps -eaf \| grep ns-slapd\"";

my $cmdstring = "ldapsearch -h $server -p $portsearch -b \"dc=Alcatel\" -z 1 \"log=\*\" log";

my @psresult = qx/$cmdstring/;

foreach $ligpsresult (@psresult) {

chomp $ligpsresult;

# if ($ligpsresult =~ m/$partdir/) { return 0; }

if ($ligpsresult =~ m/alcatel$/i) { return 0; }

}

return 1;

}