SMASER

DBCOPY

TRIPsystem
Product Documentation

End User License Agreement

All rights to this software, its documentation and logotypes of the TRIP product family and software (altogether “Software”) supplied by Smaser AG (Smaser) are exclusively owned by Smaser.

The transfer of this Software, solutions or parts thereof requires the prior written agreement of Smaser. Furthermore, the customer has the right to use licensed Software and / or process solutions supplied by Smaser to the extent specified in his contract with Smaser.

The free-to-use non-commercial version doesn’t require a prior written agreement with Smaser but such customers, organizations and/or third parties agree by using the software and / or solution of Smaser to be strongly obliged to keep all rights to this software, documentation and logotypes of the TRIP product family absolutely uninfringed and protected.

Table of Contents

INTRODUCTION 4

TRIP LOGIN AND ACCESS RIGHTS 4

SELECTION OF DATABASES TO COPY 4

FILE SYSTEM COPY DESTINATION 5

COPYING TO AN SCP/SFTP DESTINATION 5

EXCLUSIVE WRITE LOCK BEHAVIOR 6

SCRIPTED USE AND REDIRECTED/PIPED OUTPUT 6

IF SOMETHING GOES WRONG 6

USAGE EXAMPLES 7

Introduction

The DBCOPY tool is intended for use to copy databases for backup purposes on systems that have a 24/7 uptime requirement and where the databases are likely to be updated at any time and/or very frequently.

Regular file system copy-operations do not guarantee TRIP data consistency, and may, if TRIP database update operations occur at the same time, result in a corrupt copy. The DBCOPY tool copies TRIP databases while holding an exclusive TRIP write lock on them, one at a time. This ensures that the copies are guaranteed to be in a consistent state and greatly reduces the risk for corrupt backup copies.

TRIP login and access rights

The DBCOPY utility program reads database file paths from the database designs in order to ensure that it finds the correct files even in circumstances where the individual files (BAF, BIF, VIF and LOG) all reside in different locations.

Because DBCOPY needs to read the database design information as part of its operation, it requires the credentials of a TRIP user with access to the databases that are to be copied.

To copy the CONTROL database, DBCOPY needs to authenticate as the SYSTEM user. When logged in as SYSTEM and the TDBS_SUPERMAN setting is enabled, all databases can be copied in a single session.

While DBCOPY normally needs the user's password for login, this is not required if the following two preconditions both are true:

1. The TRIP user name is identical to the name of the operating system user running the DBCOPY process.
2. The profile of the TRIP user is set to "ignore password if TRIP and O/S users are the same".

You can specify both username and password on the command line:

```
dbcopy -u TRIPDBA -p 210e-433f-8887
```

Or just the username and have TRIP ask for password:

```
dbcopy -u TRIPDBA --prompt
```

Or neither, assuming that the TRIP user is the same as the OS user and is set up to allow local login without password.

Selection of databases to copy

DBCOPY can be run to copy specific databases to copy all databases. To copy specific databases, use the "`-d`" option to specify the databases. For example:

```
dbcopy -d MYDATABASE1 -d MYDATABASE2 ...
```

To copy all databases that the specified TRIP user has access to, use the "`--all`" option:

```
dbcopy --all ...
```

File system copy destination

To have DBCOPY copy database files to a directory on the local file system or one mounted to it is one of the two copy destination options available. The only requirement is that the OS user that runs the DBCOPY process has write access to said directories.

Specify the destination directory as a fully qualified path as the last of the command line arguments to DBCOPY. For example:

```
dbcopy ... /var/lib/trip-backup
```

Copying to an SCP/SFTP destination

If the backup files are supposed to be stored elsewhere and that location is not possible or convenient to reach via a file system mount, DBCOPY can be instructed to upload the database files via SCP or SFTP to a remote server.

For this, specify the URL to the destination director instead of local file system path:

```
dbcopy ... scp://example.com/home/tripbkp/data
```

DBCOPY also need to be informed of the user credentials by which it will authenticate against the remote server. These credentials can be specified as username and password via the command line using the `-U` (capital 'U') and `-P` (capital 'P') arguments:

```
dbcopy -U tripbkp -P 147d3a839c ...
```

Or as part of the URL:

```
dbcopy ... scp://tripbkp:147d3a839c@example.com/home/tripbkp/data
```

It is also possible to use a key based login instead of a having to specify password. For this, specify the port path to the SSH private key file to use with the `-i` command line argument. For example:

```
dbcopy -i ~/.ssh/id_rsa ...
```

Note that unless the remote O/S username is the same as the local O/S username, the username to authenticate as must also be specified even with key based login, either on the command line or as part of the URL.

If the remote server uses a non-standard ssh port, the port number must be specified on the command line to DBCOPY using the `-n` command line argument. E.g.:

```
dbcopy -n 22222 ...
```

Exclusive write lock behavior

The DBCOPY utility program takes an exclusive TRIP write lock on any database it is about to copy. This requires that the operating system user that runs the DBCOPY process must have write access to the database files, minimally the BAF file. Read-only access to the files will not work due to the way that the TRIP write locks operate.

If the database being copied is already write-locked by another TRIP process, DBCOPY will wait for the lock for up to 10 seconds by default before giving up. This default can be changed using the -w command line option. For example, to raise the timeout to 30 seconds:

```
dbcopy -w 30 ...
```

Scripted use and redirected/piped output

The progress status messages that dbcopy emits when copying database files works well when run directly in a terminal without redirecting or piping the output elsewhere. But the printouts will become garbled when the the output is either redirected to a file or piped to another program. That is, unless the `--batch` command line argument is used. This option tells dbcopy that it is running with its output being redirected or piped, which causes it to adjust the way the status messages are formatted.

```
dbcopy --batch ...
```

If something goes wrong

Databases may not always be possible to copy. There are a number of reasons as to why this might happen:

- The database files do not exist at the locations specified in the design. This might be because the no data has yet been stored, the database files have been manually removed, or the file paths as specified database design have errors in them.
- The O/S user running the DBCOPY tool does not have access to the database files. If the O/S user does not have write access to the BAF, it will not be possible to obtain an exclusive write lock on the database.
- The TRIP user does not have access to the database. Without any kind of access to a database, a TRIP user will not be able to see it.
- The target directory does not have sufficient space to hold copies of the files. When a file system copy somehow is interrupted, the DBCOPY utility will normally remove the partial copy so that it isn't later mistaken for a full, correct copy. When this happens via an SCP/SFTP upload, the partially uploaded file may remain on the remote server, however, so pay attention to errors reported by such uploads.
- When copying to a destination directory available on the file system, locally or mounted, DBCOPY will by default refuse to overwrite pre-existing files. To force overwriting, the "`--force`" option can be added to the command line arguments.

The DBCOPY utility will ignore errors related to the database files not being present, but still warn about such occurrences. To have DBCOPY keep silent about such errors, add the "`--skip-empty`" option to the command line parameters:

```
dbcopy --skip-empty ...
```

When more serious errors occur, DBCOPY is likely to abort processing altogether. If multiple databases are selected (by using the `--all` option), it is possible to force DBCOPY to continue anyway using the --continue option:

```
dbcopy --continue ...
```

Usage examples

Copy a single, specified database accessible by the specified TRIP user to a directory on (or mounted onto) the local file system:

```
dbcopy -u TRIPDBA --prompt -d MYDATABASE /var/lib/trip-backup
```

Copy all databases accessible by the specified TRIP user to a directory on (or mounted onto) the local file system, overwriting any existing files at the destination, skipping warnings about empty databases and allowing the copying to continue even if errors (e.g. to obtain database lock) occur.

```
dbcopy -u TRIPDBA --prompt --all --force --continue \
       /var/lib/trip-backup
```

Running on Linux or Solaris, to copy a single database via SCP to a remote server using a non-standard SSH port and a specific RSA private key file:

```
dbcopy -u TRIPDBA --prompt -d MYDATABASE -P 22222 -i ~/.ssh/tripkey \
       scp://remotuser@example.com/var/lib/trip-backup
```

Copy all databases to a local directory running DBCOPY as a user with the same name as the TRIP user, with the TRIP user being allowed to log in locally without specifying password if identically named as the O/S user. This variant is useful when running DBCOPY from an automated job:

```
dbcopy --all --force /var/lib/trip-backup
```