Raima Database Manager 12.0

DB Repair User Guide

Contents

Introduction

DBREPAIR is a suite utilities that implement a number of functions to support database maintenance in critical situations of database corruption or in cases of performance degradation. The functions of the module perform the following operations:

`dbcheck`	Database consistency check utility
`dbrepair`	Database repair utility
`keybuild`	Key file build utility
`dbdefrag`	Database defragmentation utility
`dchain`	Delete chain sort utility
`dbcluster`	Database set optimization utility

There are two types of database corruptions: physical and structural. Physical database corruption is caused by writing invalid data directly into a database file or through a corrupted cache page. The former can happen through an occasional direct modification of the database files from other applications. The latter can only be caused by an error in the direct-linked application or by a bug in the runtime engine. As a result, some fields in the database records contain garbage data.

In general, the DBREPAIR utility cannot restore the value of the corrupted data fields. But if the fields are the database pointers used for a set relationship, the utility can attempt to restore these values in those fields using redundant pointers in doubly-linked lists. We say that a database pointer, which is a value of the DB_ADDR type, is physically corrupted if it is not valid for the particular database. A DB_ADDR value is valid as a database pointer if its `fileno` component is a valid file number and its `slotno` component is a valid slot number for this file. If a pointer is a set pointer the `fileno` component additionally must be consistent with the set definition in the database schema.

Structural corruption is caused by inconsistency in several associated pointers that represent owner-member relationships between database records.

Utility Descriptions

This section contains descriptions of the Db Repair utilities in alphabetical order. Each page header contains the name of the utility for easy reference. The Prototype section contains the command-line syntax showing all possible parameters and options. The Description section explains the purpose of the utility and any command-line parameters. The Options section explains each command-line option.

dbrepair

Database repair utility

Prototype

```
dbrepair [-h] [-B] [-V] [-q|[-stdout file]] [-stderr file] [-tfs type]
         [-key key] [-docroot path] [-c] [-v] [-locale locale] dbname
```

Description

This utility performs the consistency check of the *db_namespec* database and repairs the database if corruption is detected. The utility extracts the parameters for the call from the values of the command options.

Options

Specifies the server port (TCP/IP) or name (non-TCP-IP) for connections. The default port is 21553 and the default name is "21553". If a number is specified it will be used for the TCP/IP port and the server name. If a string or invalid port number is specified it will be used as the server name and the TCP/IP transport will not be initialized.

`-h | -?`	Display this usage information
`-B`	Do not display the banner
`-V`	Display the version information
`-q`	Quiet mode. No information will be displayed
`-stdout` *filespec*	Redirect `stdout` to the specified *filespec*
`-stderr` *filespec*	Redirect `stderr` to the specified *filespec*
`-tfs` *type*	Specify the TFS *type* to use: `tfss | standalone | s` — Standalone TFS type `tfsr | rpc | r` — Client/Server TFS type `tfst | tfs | t` — Direct-Link TFS type (Default)
`-key` *enckey*	Specify the encryption key for the database. The *enckey* is specified as: `[algorithm:]passcode`. The valid algorithm are: xor — XOR encryption key `aes128` — 128 bit AES encryption key `aes192` — 192 bit AES encryption key `aes256` — 256 bit AES encryption key
`-docroot` *path*	Document root under which to place database files.
`-trans` *trans_list*	Specifies a comma separated list of transports to listen on. Options are "tcp" and "shm" (default is "shm,tcp")
`-p` *port/name*	Specifies the server port (TCP/IP) or name (non-TCP-IP) for connections. The default port is 21553 and the default name is "21553". If a number is specified it will be used for the TCP/IP port and the server name. If a string or invalid port number is specified it will be used as the server name and the TCP/IP transport will not be initialized.
`-v`	Verbose output.
`-start`	Start the server (run as a daemon on Unix)
`-stop`	Stop the server

`-query`	Query the execution status of the server
`-rdonly`	Prohibit any operation that would modify the contents of the document root.
`-nodisk`	Don't use any disk I/O.
`-notify`	Enable HA notification
`-notfslisten`	Do not allow direct connection to TFS
`-nopageref`	Disable *page referencing*[1] read optimization
Windows Only	
`-install` *`path`*	Install the server as a service at the specified path.
`-uninstall`	Uninstall the server as a service.
`-serviceuser`	User account to install service as (only valid with `-install` option)
`-servicepw`	User account password to install service as (only valid with `-install` option and required with `-serviceuser` option)
`-c`	Number of Cache pages (default:10)
`-locale`	Specify a locale to use for ordering string values
`db_namespec`	Database name.

See Also

Database Name Specification (db_namespec)

Document Root (DOCROOT)

[1]In a TFSR configuration - where the application runs in a separate process from the TFS (which would be running in a tfserver or other application), the system normally attempts to determine if the application is on the same physical machine as the TFS. If it is determined that the two processes are on the same machine, the code will use a shortcut - those pages that will be read from the database files and sent over to the application via TCP/IP or other connection are instead read directly from the disk by the user application. This is all handled internally by the software - no separate options are required. Some users, however may not wish for this to take place because of file permission issues or other reasons. In that case, this option can be specified and doing so will disable this shortcut - thus acting as if the user application and the TFS are on different machines and send all pages through the connection between the two.

dbcheck

Database consistency check utility

Prototype

```
dbcheck [-h] [-B] [-V] [-q|[-stdout file]] [-stderr file] [-tfs type]
        [-key key] [-docroot path] [-s] [-k] [-dk] [-kd] [-v] [-dv] [-vd]
        [-b] [-db] [-a] [-openmode mode] [-nk] [-nv] [-nb] [-qd] [-qk]
        [-r #] [-p #] [-f #] [-t] [-c] [-l] [-locale locale]
        db_namespec [dbfiles]
```

Description

This utility checks the consistency of all database files in database `db_namespec`. If any database files (dbfile) are specified, only those files are checked. Otherwise, all files in the database are checked.

The `dbcheck` utility checks database consistency by validating the position of each record occurrence and checking the integrity of the delete chains. Options perform more complete consistency checks. Use of these options slows down `dbcheck` execution.

With the -s option, the `dbcheck` utility can perform set consistency checking. Set membership consistency verifies the following:

1. Member and owner record types are valid
2. Membership count is correct
3. Doubly linked lists are properly formed

The -k option causes `dbcheck` to verify the B-tree structure of the key files. This ensures that each node (except the root node) is at least half full, that the number of filled slots is correct, that the key slots on each node are properly sorted, and that each leaf node is at the correct level in the B-tree. Note that if the -dk option does not result in any errors, it is very unlikely that the -k option will result in any inconsistencies.

Key file structure consistency checks performed (optionally) verify that:

1. All B-tree pages are either in the B-tree or are deleted
2. All B-tree null child pointers are at the same tree level
3. B-tree key order is correct
4. Duplicate keys are correct

The -dk option validates the existence of the key values associated with each record and key field in the data files. The -kd option will read, for each key in a key file, the associated record and check to ensure that the key's data field contents matches that stored in the key file.

Inconsistencies are reported with a message indicating the nature of the inconsistency and the file and location of the offending record or key. If the -t option is specified, a trace back of the B-tree is printed when a key file inconsistency is detected.

The -v option causes `dbcheck` to verify the structure of varchar files. This ensures that each varchar is less than the maximum size, that the varchar pointers point to valid locations, and varchar file chunk points to a valid varchar field.

Options

`-h | -?`	Display this usage information
`-B`	Do not display the banner
`-V`	Display the version information
`-q`	Quiet mode. No information will be displayed
`-stdout` *`filespec`*	Redirect `stdout` to the specified *filespec*
`-stderr` *`filespec`*	Redirect `stderr` to the specified *filespec*
`-tfs` *`type`*	Specify the TFS *type* to use: `tfss | standalone | s` — Standalone TFS type `tfsr | rpc | r` — Client/Server TFS type `tfst | tfs | t` — Direct-Link TFS type (Default)
`-key` *`enckey`*	Specify the encryption key for the database. The *enckey* is specified as: `[algorithm:] passcode`. The valid algorithm are: xor — XOR encryption key `aes128` — 128 bit AES encryption key `aes192` — 192 bit AES encryption key `aes256` — 256 bit AES encryption key
`-docroot` *`path`*	Document root under which to place database files.
`-s`	Performs a complete consistency check of sets.
`-k`	Performs B-tree structure consistency check. This checks to ensure that the key file adheres to all of the rules for a B-tree. If -dk is also specified, this check is unnecessary.
`-dk`	Checks the existence of each key from each record occurrence. While scanning a data file, a `d_keyfind` is performed on each key to ensure that it exists.
`-kd`	Checks the existence of each record from each key. While scanning a key file, a `d_recread` function call is performed and the associated key field value is checked against the key to ensure that it matches.
`-v`	Perform checks on varchar files.
`-dv`	Checks the existence of each varchar chunk referenced from a varchar field while scanning a data file.
`-vd`	Checks the validity of the owner pointer for each varchar chunk in a varchar file.
`-b`	Perform BLOB file structure consistency check.
`-db`	Checks the existence of each BLOB from each record occurrence.
`-a`	All. Performs all of the above consistency checks.
`-openmode` *`mode`*	Specify the open mode to use for the database (default 's' using ROT).
`-nk`	Disables key file checking, -k, -dk and -kd.
`-nv`	Disables varchar checking, -v, -dv and -vd.
`-nb`	Disables blob checking, -b and -db.
`-qd`	Quick dchain checks (only report one dchain error per file).
`-qk`	Quick key checks (only report one error per key file).
`-r #`	Reports every # percentage completed to stderr.
`-p #`	Sets to # the number of pages in the RDM cache for use by `dbcheck`. For example, `-p 128` specifies that `dbcheck` is to allocate 128 pages for the cache. The default is 64 pages. If an insufficient memory error (S_NOMEMORY) occurs when starting `dbcheck`, use the `-p` option to specify a value less than 64.
`-f #`	Maximum number of open files allowed to have open at once.
`-t`	Prints a trace back of the B-tree when a key file inconsistency is detected.
`-c`	Prints a count of objects scanned in the check.

`-l`	Log errors to repair database.
`-locale` *`locale`*	Specify a *locale* to use for ordering string values.
`db_namespec`	Database name.

See Also

Database Name Specification (db_namespec)

Document Root (DOCROOT)

dbcluster

Database set optimization utility

Prototype

```
dbcluster [-h] [-B] [-V] [-q|[-stdout file]] [-stderr file] [-tfs type]
          [-key key] [-docroot path] [-locale locale] [-v]
          db_namespec [@datfile|setname(s)]
```

Description

The `dbcluster` utility relocates all members of the selected sets contiguously in the database storage files.

A database contains one or more data files. Each data file contains database pages of a specific size (size can be set by the user), with each page containing as many records (slots) as will fit on a page. The data files can contain multiple owner and member record types. It is easy to conceive that during normal database activity many different types of records could be inter-mixed during inserts. As this occurs and the database grows large, performance can degrade due to set navigation having to traverse a set through multiple pages. `dbcluster` can help by placing related set members into contiguous record slots.

It is not always possible to optimize every set defined in a database. If a record is a member in multiple sets it is only possible to optimize one of the sets. You can either explicitly specify the set to optimize or the utility will choose the first set defined in the database.

Procedures

1. Run `dbcheck` on your database and ensure that the database is valid.
2. Backup your database.
3. Run the `dbcluster` utility.

Options

`-h | -?`	Display this usage information
`-B`	Do not display the banner
`-V`	Display the version information
`-q`	Quiet mode. No information will be displayed
`-stdout` *`filespec`*	Redirect `stdout` to the specified *filespec*
`-stderr` *`filespec`*	Redirect `stderr` to the specified *filespec*
`-tfs` *`type`*	Specify the TFS *type* to use: `tfss | standalone | s` — Standalone TFS type `tfsr | rpc | r` — Client/Server TFS type `tfst | tfs | t` — Direct-Link TFS type (Default)
`-key` *`enckey`*	Specify the encryption key for the database. The *enckey* is specified as: `[algorithm:]passcode`. The valid algorithm are: xor — XOR encryption key `aes128` — 128 bit AES encryption key `aes192` — 192 bit AES encryption key

	`aes256`	256 bit AES encryption key
`-docroot` *`path`*	Document root under which to place database files.	
`-locale locale`	Specify a locale to use for collating string values	
`-v`	Verbose - shows detailed output of the operations performed on the database.	
`db_namespec`	Specifies the database to optimize	
`datfile`	A file that contains a list of set names to optimize	
`setname(s)`	A list of sets to optimize	

See Also

Database Name Specification (db_namespec)

Document Root (DOCROOT)

dbdefrag

Database defragmentation utility

Prototype

```
dbdefrag [-h] [-B] [-V] [-q|[-stdout file]] [-stderr file] [-tfs type]
         [-key key] [-docroot path] [-locale locale] [-v] db_namespec
```

Description

Each database consists of one or more data files. Each of these data files contains database pages which in turn consist of database slots (records).

Normally, whenever a record is deleted from the database, it is marked for reuse and put on the delete chain, but not physically removed from the file. These free (deleted) records can occupy a substantial amount of disk space, especially if a database has had numerous records inserted and subsequently many of them marked as deleted.

The `dbdefrag` utility moves all valid records to the front of a data file and all deleted slots to the end of the data file. The data file is then truncated in order to reclaim disk space.

The defragmentation takes place in the following operations.

1. A temporary database is created by `dbdefrag` to store the addresses of the deleted slots
2. The records are moved to the front of the database and the files are truncated if warranted.
3. The references to set owners, set members and database address fields, if used, are updated.
4. New key files are created based upon the new data files.

Options

`-h | -?`	Display this usage information
`-B`	Do not display the banner
`-V`	Display the version information
`-q`	Quiet mode. No information will be displayed
`-stdout` *`filespec`*	Redirect `stdout` to the specified *filespec*
`-stderr` *`filespec`*	Redirect `stderr` to the specified *filespec*
`-tfs` *`type`*	Specify the TFS *type* to use: `tfss | standalone | s` — Standalone TFS type `tfsr | rpc | r` — Client/Server TFS type `tfst | tfs | t` — Direct-Link TFS type (Default)
`-key` *`enckey`*	Specify the encryption key for the database. The *enckey* is specified as: `[algorithm:]passcode`. The valid algorithm are: xor — XOR encryption key `aes128` — 128 bit AES encryption key `aes192` — 192 bit AES encryption key `aes256` — 256 bit AES encryption key
`-docroot` *`path`*	Document root under which to place database files.
`-locale` *`locale`*	Specify a locale to use for collating string values

`-v`	Verbose - shows detailed output of the operations performed on the database.
`db_namespec`	Database name.

See Also

Database Name Specification (db_namespec)

Document Root (DOCROOT)

dchain

Delete chain sort utility

Prototype

```
dchain [-h] [-B] [-V] [-q|[-stdout file]] [-stderr file] [-tfs type]
       [-key key] [-docroot path] [-v] dbname [filename ...]
```

Description

The `dchain` utility sorts the deleted record slots on the delete chains of the listed data files ([*dbfile*]...) from database `db_namespec` in database address order. If no files are listed, all data files in database `db_namespec` will be processed. This utility does not sort key file delete chains.

The purpose of this utility is to increase the probability that newly created records in the same file will be placed close together. The extent of any performance improvement, however, will depend upon the application. The most likely time to use this utility is after a periodic purge of a particular set of record types, many of which were entered together; thus there would be many contiguous record slots on the delete chain.

Options

`-h | -?`	Display this usage information	
`-B`	Do not display the banner	
`-V`	Display the version information	
`-q`	Quiet mode. No information will be displayed	
`-stdout` *`filespec`*	Redirect `stdout` to the specified *filespec*	
`-stderr` *`filespec`*	Redirect `stderr` to the specified *filespec*	
`-tfs` *`type`*	Specify the TFS *type* to use:	
	`tfss | standalone | s`	Standalone TFS type
	`tfsr | rpc | r`	Client/Server TFS type
	`tfst | tfs | t`	Direct-Link TFS type (Default)
`-key` *`enckey`*	Specify the encryption key for the database. The *enckey* is specified as: `[algorithm:] passcode`. The valid algorithm are:	
	xor	XOR encryption key
	`aes128`	128 bit AES encryption key
	`aes192`	192 bit AES encryption key
	`aes256`	256 bit AES encryption key
`-docroot` *`path`*	Document root under which to place database files.	
`-v`	Display detailed output about all operations (to stdout).	
`db_namespec`	Database name.	

See Also

Database Name Specification (db_namespec)

Document Root (DOCROOT)

keybuild

Key file build utility

Prototype

```
keybuild [-h] [-B] [-V] [-q|[-stdout file]] [-stderr file] [-tfs type]
         [-key key] [-docroot path] [-f #] [-locale locale] [-p #] dbname
```

Description

The `keybuild` utility rebuilds all key files for the database `db_namespec`. Rebuilding key files is a two-step process. First, the file is reinitialized. Then, each record is sequentially read from each data file record, and each key file is re-created from the record contents.

This utility can be used to re-create the key files when `dbcheck` reports a database inconsistency. The utility can also construct new key files after you have added or removed key attributes from fields in your DDL specification. For example, if you make an existing key field a non-key, and change a non-key to a key field in your DDL, you can run `keybuild` to rebuild the key files for the new schema. You can also use `keybuild` to reassign key fields to different key files.

Options

`-h | -?`	Display this usage information
`-B`	Do not display the banner
`-V`	Display the version information
`-q`	Quiet mode. No information will be displayed
`-stdout` *`filespec`*	Redirect `stdout` to the specified *filespec*
`-stderr` *`filespec`*	Redirect `stderr` to the specified *filespec*
`-tfs` *`type`*	Specify the TFS *type* to use: `tfss | standalone | s` Standalone TFS type `tfsr | rpc | r` Client/Server TFS type `tfst | tfs | t` Direct-Link TFS type (Default)
`-key` *`enckey`*	Specify the encryption key for the database. The *enckey* is specified as: `[algorithm:] passcode`. The valid algorithm are: xor XOR encryption key `aes128` 128 bit AES encryption key `aes192` 192 bit AES encryption key `aes256` 256 bit AES encryption key
`-docroot` *`path`*	Document root under which to place database files.
`-locale` *`locale`*	Specify a locale to use for sorting string values.
`-f` *`N`*	Report progress to `stdout` every N records. Default is 1000. Set to 0 for no report.
`-p` *`N`*	The number of cache pages to use. Default is 1000.
`db_namespec`	Database name.

See Also

Database Name Specification (db_namespec)

Document Root (DOCROOT)

Appendix

Database Name Specification (db_namespec)

The database name specification uses a Uniform Resource Identifier (URI) format to identify the TFS and database name. The syntax for the naming convention is:

```
dbname_spec:
        ‹tfs_spec›db_name

tfs_spec:
        'tfs:///'
    |   'tfs-tcp://' tcp_addr '/'
    |   'tfs-shm://' shm_name '/'

tcp_addr:
        {tcp_id | hostname}‹':' portnum›
    |   ':' portnum

tcp_id:
        ‹[› <IPv4 address> ‹]›
    |   '[' <IPv6 address> ']'
```

The elements above enclosed with chevron characters (‹ and ›) are optional elements in the specification grammar.

Supported Transport Types (tfs_spec)

`tfs-shm://`*`shm_name`*`/`	The shared-memory communications transport is the default transport a RDM client running on the same machine as the TFS.
`tfs-tcp://`*`tcp_addr`*`/`	RDM supports IPv4 and IPv6 TCP/IP connections. Since the colon character (':') is used to separate the hostname and port parts of a TCP/IP address and the numeric notation for IPv6 addresses also contains colons, square brackets are required around an IPv6 address in numeric notation. Square brackets can also be used around IPv4 addresses in numeric notation, but are not required.
`tfs:///`	Uses the default transport type available. The RDM client will first attempt to connect to the TFS using the shared-memory communication transport followed by the TCP/IP communication transport.

Default TFS Addresses

Specifies the server port (TCP/IP) or name (non-TCP-IP) for connections. The default port is 21553 and the default name is RDM_21553. If a number is specified it will be used for the TCP/IP port and the server name will be RDM_port. If a string (non-number) is specified it will be used as the server name and the TCP/IP transport will not be initialized.

`21553`	Default port number (used in default shared-memory name and default TCP address below)
`21553`	Default shared-memory name.
`localhost:21553`	Default TCP host and port address

The `tfs_spec` should not be used for TFS types: TFSS and TFST. Only TFSR currently supports the `tfs_spec`.

Database Name Specifications

db_namespec	Description
`sales`	"sales" on default TFS using default transport.
`tfs:///sales`	"sales" on default TFS using default transport.
`tfs-tcp://www.raima.com/sales`	"sales" using TCP/IP with the default port number at hostname "www.raima.com"
`tfs-shm://partition-01/sales`	"sales" using shared-memory to TFS named "partition-01"
`tfs-tcp://[::1]:1530/sales`	"sales" using TCP/IP IPv6 using the loopback address on port 1530
`tfs-tcp://[fe80::40da:bf3f:ae9f:fe87]:2000/sales`	"sales" using TCP/IP IPv6 to a specified machine on port 2000.
`tfs-tcp://192.168.101.139:2000/sales`	"sales" using TCP/IP IPv4 to a specified machine on port 2000.

Union Database Name Specifications

A database union is a unified view of the data in more than one identically structured database. It makes the multiple databases appear as one. A union of multiple databases differs from having multiple databases open in that:

- unioned databases must have identical dictionaries (DBD files)
- the union is viewed as one database, using one database number
- it is read-only

Database unions are intended to be used with distributed databases or database mirrors to create a single, merged view of data that is owned and updated in separate locations or by separate entities. Database specifications are separated by a vertical bar (|) in the `db_namespec`.

Note that a database union is a union of different instances of the same database schema (i.e., definition) contained on separate TFSs. This is not to be confused with the standard SQL **union** of **select** statements operation.

Example Union Database Open Specifications

```
d_open("tfs-shm://partition-01/sales|tfs-tcp://www.raima.com/sales", "r", task);
rsqlOpenDB(hdbc, "tfs-tcp://localhost:21553/sales|tfs-tcp://www.raima.com/sales:1530", "r");
open nsfawards as union of "tfs-shm://partiion-01", "tfs-tcp://www.raima.com:1650":
```

Document Root (DOCROOT)

The document root (DOCROOT) is a directory (a folder) which is designated for holding databases available to a Transaction File Server (TFS). The concept is similar to a web server document root for storing web pages. Important notes about the TFS are:

- A TFS has exclusive access to a document root. Multiple simultaneous TFS access to a DOCROOT is not allowed;
- Within the domain of one TFS, no files outside the DOCROOT can be accessed;
- A database, by default, will be a sub-directory of the DOCROOT of the same name;
- The DOCROOT path cannot be the root directory of a files system.

The default Direct-Link (TFST) includes an embedded `tfserver` in the process which means that only one Direct-Link application process at a time can access the document root. Multiple process access to a document root requires the use of the Client-Server (TFSR) configuration.

Shared Memory Transport:

The shared memory transport is an alternative to TCP/IP communication when both the server and the client communicating with it reside on the same machine. Instead of creating a socket for communications, a simple shared memory buffer is used.

On Windows this uses a file mapping object that is backed by the system paging file.

On non-Windows systems this uses a memory mapped file which will be created in '/tmp' by default (this can be changed using the RDMTEMP environment variable if /tmp is not desired - but it must be the same location for both the server process and each client accessing that server).

Data being sent between the two processes will be copied into this buffer and then the other side will be notified and it will copy the data out. In order to implement this efficiently, certain atomic functions are required and they are not present on every system. Those systems that do not support these atomic functions will not have the shared memory transport available to them.

The benefit of the shared memory transport for same machine communication to a server is performance. The share memory transport is faster than TCP/IP as there is less overhead to go through to get data back and forth. Both shared memory and TCP/IP can be available at the same time so it is possible to have remote clients communicate with the server via TCP/IP and local clients communicate via shared memory.

When communicating via TCP/IP, a hostname and a port are required to identify which server to communicate with. When using shared memory, the hostname is not required (as the client will always be on the same machine as the server), and instead of a port, a name is used. This name is string of characters and needs to follow the guidelines for a filename on the system being used (i.e. don't use a directory character ('/' or '\') in the name).