NAME

dachs - data publishing infrastructure for the Virtual Observatory (VO)

SYNOPSIS

dachs [global-options] <subcommand> [options] function-argument ...

DESCRIPTION

dachs provides support for data ingestion and publishing, for metadata handling, and for a variety of VO protocols and standards, e.g. the Table Access Protocol (TAP) or the Simple Cone Search (SCS).

There are numerous sub-commands covering the various tasks (importing, controlling the server, running tests, etc).

Subcommand names can be abbreviated to the shortest unique prefix.

A centeral concept of DaCHS is the Resource Descriptor (RD), and XML description of a data collection including metadata, ingestion rules, service definitions, and regression tests. They are usually referenced through their RD ids, which are the relative paths from DaCHS' inputs directory to the file containing the RD, with the conventional extension **.rd** stripped. For instance, in a default install, the file **/var/gavo/inputs/myrsc/q.rd** would have **myrsc/q** as RD id.

Most commands dealing with RD ids will also pick up RDs if referenced by path; in the example above, if you are in **/var/gavo/inputs/myrsc**, you could also reference the RD as either **q** or **q.rd**.

Several commands take references to RD elements (table definitions, exec items, direct grammar, etc). These consist of an RD id as just discussed, a hash mark, and the XML id of the target element. Tables have an id automatically, for other elements you may have to add an artificial id.

GLOBAL OPTIONS

Global options are given before the subcommand name.

--debug
produce debug info as appropriate

--enable-pdb
run pdb on all errors

-h, **--help**
show this help message and exit

--hints
if there are hints on an error, display them

--profile-to=*PROFILEPATH*
enable profiling and write a profile to PROFILEPATH

--suppress-log
suppress logging of exceptions to the dachs-specific log files

--version
shows the versions of the software, of the database schema expected by the software and of the database schema actually on disk (if the latter two disagree, run dachs upgrade).

THE ADMIN SUBCOMMAND

Synopsis:

dachs admin [-h] subsubfunction [subfunction-arguments ...]

This is a somewhat random collection of commands related to administering a data center. In particular, this is where you create and edit accounts.

subsubcommands can be abbreviated as long as the abbreviation is unique. For instance **dachs adm xsd** will do an XSD validation.

For more information on the subsubfunctions, pass a **-h** flag.

Subsubcommands

- **addtogroup user group** -- adds a user to a group
- **adduser user password [remarks]** -- add a user/password pair to the DaCHS user table. Note that as of DaCHS 1.0 the password is stored in clear text and also transmitted in clear text since DaCHS only supports HTTP basic authentication. Do not use valuable passwords here
- **changeuser user password [remarks]** -- change remarks and/or password for a DC user. See adduser for details.
- **cleantap** -- remove expired Universal Worker Service (UWS) jobs
- **delfromgroup user group** -- remove a user from a group

- **deluser user** -- remove a DaCHS user from the user table.

- **dumpDF path** -- Dumps the source of a file included with the DaCHS distribution. The argument is a package resource path (like /inputs/__system__/scs.rd); for system RDs, the special //rd-id syntax is supported.

- **execute exec-id** -- Execute the contents of an RD execute element. You must give that element an explicit id in order to make this work; then exec-id is rd-id#exec-id

- **hipsfill [-d HIPSDIR] svcId** -- complete the properties file in HIPSDIR with metadata from the service referenced by svcId (e.g., q#hips).

- **hipsgen dataId minOrder** -- Write a Hipsgen parameter file for making a HiPS out of the data imported by dataId (e.g., q#import) to stdout. minOrder is the minimal HEALPix order Hipsgen should generate (low orders are computationally expensive!)

- **hashPassword pw** -- hash a password. adduser and friends already hash themselves, so this is likely only useful for the [web]adminPasswd setting.

- **indexStatements tableRef** -- show the statements to create the indices on a table. The tableRef has the format RD-id#table-id; it is *not* a database table reference.

- **listusers** -- dump the user table

- **makedelrec ivoid** -- create a registry record of ivoid saying the resource is removed. This is only necessary when recovering from disasters that made you lose records from DaCHS' internal tables.

- **suggestucds tableId** -- Make suggestions for UCDs of the columns of the referenced table (rd-id#table-id format) not having one. This is based on their descriptions and uses a GAVO web service.

- **tapabort jobId helpMsg** -- manually abort a TAP job and return helpMsg to the requesting user as error message.

- **updateTAPSchema** -- Update the TAP_SCHEMA metadata for all RDs mentioned in TAP_SCHEMA.

- **xsdValidate path** -- Validate a file against built-in VO schemas and with built-in schema validator.

THE CONFIG SUBCOMMAND

Synopsis:

dachs config [section-name] config-key

This outputs values of DaCHS' configuration to stdout. section-name defaults to general. This is most commonly used to make external components aware of DaCHS' file locations, e.g., through **inputs_dir=$(dachs config inputsDir)**.

See the operator's guide for a documentation on DaCHS' configuration options.

THE DATAPACK SUBCOMMAND

Synposis:

dachs datapack [-h] {create,load} ...

Management of full DaCHS resources in the frictionless data package format. Note that while DaCHS-created data packages should work as normal data packages, DaCHS can only (automatically) load data packages generated by DaCHS itself -- there simply is not enough metadata in generic data packages to produce usable resources.

Subsubcommands

datapack create

Synopsis:

dachs datapack create [-h] id dest

Positional arguments:

id Absolute RD id to produce a datapack for

dest

Name of a zip file to dump to

The command creates a data package containing the RD, a README in the resource directory if it exists, auxiliary files (like a booster grammar or a custom core) the RD may have, and all data files that the sources in the RD yield; for collecting these files, **sources** elements with **ignoreSources** children reading from the database are ignored, because that would make behaviour a bit too unpredictable.

To include further files, you can use the datapack-extrafiles property on the RD. This must

contain a json sequence literal with resdir-relative shell patterns that should be included as well. Non-existing paths and directories in this list are silently ignored. You cannot include files outside of the resource directory in a data package.

datapack load

Synopsis:

dachs datapack load [-h] [-t] [--force] source

Positional argument:

source

Name of a DaCHS-produced data package zip file.

Optional arguments:

-t, **--no-test**

Do not run tests after importing.

--force

If the resdir the package declares already exists, remove it before unpacking (rather than bailing out.

This unpacks a DaCHS-produced data package, imports it, and runs tests on it (which you will want to suppress if you do not have a DaCHS server running locally).

THE DROP SUBCOMMAND

Synopsis:

drop [-h] [-s] [--all] rd-id [dd-id ...]

This is the reverse of import: Tables created by a **dachs imp** with identical arguments are being torn down by **dachs drop**. This will not work reliably if the RD has been modified between the imp and the drop, in particular if the RD has been deleted. In such situations, you can use the **-f** flag, which unconditionally tears down everything DaCHS has recorded as coming from the referenced RD.

Arguments

rd-id

RD path or ID to drop

dd-id

optional data descriptor (DD) ID(s) if you do not want to drop the entire RD; note that no service publications will be undone if you give DD IDs

Options

--all

drop all DDs in the RD, not only the auto ones (overrides manual selection)

-s, **--system**

drop tables even if they are system tables

THE DUMP SUBCOMMAND

Synopsis:

dachs dump [-h] {load,create,ls} ...

This is an interface to dumping database tables and inspecting and restoring the generated dumps.

This is mainly intended for small to medium tables that are just kept in the database, e.g., DaCHS' administrative tables and the like. For normal user tables, built from science data, doing re-imports is the recommended way to deal with data loss.

In particular, this command is *not* designed (at this point) for really large tables. For technical reasons (that could be overcome), currently the individual dumps are kept in memory during creation (but not during loading).

Before loading, the target tables are dropped if they are already present; note that this will also drop any views they might be part of, as well as any rows that have foreign keys on this table. After loading, indices and primary keys on the restored tables will be recreated, but no scripts or similar are run. Hence, you will have to manually re-create any dependent resources after a restore.

This also means that, for example, restoring a table with products without also restoring //products#products will lose the products that previously were in the restored table from the products table.

Subsubcommands

dump create

Synopsis:

dachs dump create [options] dumpFile ids [ids ...]

Dump one or more tables to DaCHS' dump format. When you pass in RD ids, all onDisk-tables defined in the RD will be dumped.

Positional arguments:

dumpFile

Name of a file to write the dump to; use - to dump to stderr.

ids ids of table definitions (as in myres/q#main) or RDs to dump.

Options:

--text

write dumps in text format. You need that when objects in the tables have no binary representation. At least in 2020, that is true for all pgsphere objects.

dump load

Synopsis:

dachs dump load [-h] source

Restore table(s) from a file created by the **dump create** subcommand before

Positional argument:

source

File to restore from. Use - to restore from stdin.

dump ls

Synopsis:

dachs dump ls [-h] source

List tables and dump metadata from a DaCHS dump.

Positional arguments:

source
source file to list

THE GENCOL SUBCOMMAND

Synopsis:

dachs gencol [-h] [-f {fits,vot,viz}] FILE

The gencol subcommand quickly generates a set of basic column elements for inclusion in RDs. It is particularly convenient together with **dachs start**.

This currently understands FITS binary tables, VOTables and Vizier-Style byte-by-byte descriptions (the latter somewhat haphazardly). It assumes there is only one table in each file (and will ignore others in the FITS and VizieR case).

It will extract what metadata there is and emit formatted RD XML to stdout; this should work just fine for cut-and-paste into <table> elements in RD. Note that usually, important metadata will be missing (e.g., UCDs for FITS and VizieR), and usually data providers are not terribly careful when writing metadata. Hence, you should definitely thoroughly review the elements created before using them in a public service.

The program tries to guess the applicable parser from a file name (e.g., README or anything ending in .txt will be treated as Vizier-like). Where this fails, use the -f option.

For Vizier-like descriptions, the program also dumps **colDefs** material for use in columnGrammar-s.

Argument

FILE
Source file to generate the columns from. This can be a FITS, a VOTable, or Vizier Column-by-column description, and DaCHS will guess based on the extension. Use the format option if it guesses wrong.

Options

-h, **--help**
show this help message and exit

-f *FMT*, **--format** *FMT*
Format of the input table: FITS binary (**fits**), VOTable (**vot**), or Vizier byte-by-byte (**viz**)

THE IMPORT SUBCOMMAND

Synopsis:

dachs import [options] rd-name [data-id]

This subcommand is used to ingest data described by an RD. For special applications, ingestion can be restricted to specific data items within an RD.

Options

-h, **--help**
show this help message and exit

-n, **--updateRows**
Deprecated. Use updating data items instead.

-d, **--dumpRows**
Dump raw rows as they are emitted by the grammar.

-D, **--dumpIngestees**
Dump processed rows as emitted by the row makers.

-R, **--redoIndex**
Drop indices before updating a table and recreate them when done

-m, **--meta-only**
just update table meta (privileges, column descriptions,...). This will also update the tables' obscore clauses, but you will have to **dachs imp //obscore** to update the obscore view itself in this case.

-I, **--meta-and-index**
do not import, but update table meta (privileges, column descriptions,...) and recreate the indices

-u, **--update**
update mode -- don't drop tables before writing.

-s, **--system**

(re-)create system tables, too

-v, **--verbose**
talk a lot while working

-r, **--reckless**
Do not validate rows before ingestion

-M *MAX*, **--stop-after=***MAX*
Stop after having parsed MAX rows

-b *N*, **--batch-size=***N*
deliver N rows at a time to the database.

-c, **--continue-bad**
do not bail out after an error, just skip the current source and continue with the next one.

-L, **--commit-after-meta**
commit the importing transaction after updating the meta tables. Use this when loading large (hence -L) data sets to avoid keeping a lock on the meta tables for the duration of the input, i.e., potentially days. The price is that users will see empty tables during the import.

--hide-adql
Interpret adql=True on tables as adql=hidden (this is for fallback mirrors)

THE INFO SUBCOMMAND

Synopsis:

dachs info [-h] table-id

This displays column statistics about the table referred to in the argument (which must be a fully qualified table name resolvable by the database system).

Argument

table-id
table ID (of the form rdId#tableId)

THE INIT SUBCOMMAND

Synopsis:

dachs init [-h] [-d DSN] [--nodb]

This initialises DaCHS' file system and database environment. Calling **dachs init** on an existing site should not damage anything. It might, however, fix things if, for instance, permissions on some directories went funny.

Options

-d <*DSN*>, **--dsn** <*DSN*>
data source name (DSN) to use to connect to the future DaCHS database; the DSN must let DaCHS connect to the database as an administrator; dbname, host, and port get copied to the profile, if given; this is not required if the calling user has superuser privileges on a database running on localhost. Otherwise, the DSN has to look like "host=example.org dbname=mydb user=superuser password=secret".

--nodb
inhibit initialization of the database (you may want to use this when refreshing the file system hierarchy)

THE LIMITS SUBCOMMAND

Synopsis:

dachs limits [-h] [-t] [-s P] [-d] itemId [itemId ...]

This creates or updates DaCHS' estimates of various metadata of RDs or tables, in particular the STC coverage, the size of the tables and, unless **-t** is given, the column statistics. This may take a long time for large tables, in which case you may use the **--sample-percent** option, which makes DaCHS only look at a subset of the full data.

When running **dachs limits** on an RD, it will skip views under the assumption that in the typical case, column metadata for the interesting columns will already have been obtained for the source tables. For views for which this is assumption is wrong, set the forceStats property to True.

You should in general run **dachs limits** before publishing a resource.

Arguments

itemId
either an RD id or a table reference in the from rd-id#table-id. A single ALL will expand to all

RDs that already have limits-obtained metadata.

Options

-t, **--tables-only**
Only acquire table/resource-level metadata (rather than column metadata, which usually takes a lot longer).

-s *P*, **--sample-percent** *P*
Only look at P percent of the table to determine min/max/mean.

-d, **--dump**
Do not obtain statistics but rather dump the results of the last run

THE MKBOOST SUBCOMMAND

Synopsis:

dachs mkboost [option] <id-of-directGrammar>

This writes a C source skeleton for using the direct grammar referenced to fill a database table. See the *Guide to Write Booster Grammars* in the DaCHS documentation for how to use this command.

Options

-b, **--binary**
generate a skeleton for a binary parser

-s <*SPLITTER*>, **--splitter=**<*SPLITTER*>
generate a split skeleton with split string <SPLITTER>

THE MKRD SUBCOMMAND

Synopsis:

dachs mkrd [option] sample

Rudimentary support for generating RDs from data. This was probably a bad idea. Better use dachs start.

Options

-f <*SRCFORM*>, **--format=**<*SRCFORM*>
source format: FITS or VOT; default: detected from file name

-t <*TABLENAME*>, **--table-name=**<*TABLENAME*>
name of the generated table

THE PUBLISH SUBCOMMAND

Synopsis:

dachs pub [-h] [-a] [-m] [-k] [-u] rd [rd ...]

This marks data and/or services contained in an RD as published; this will make them displayed in DaCHS' portal page or pushed to the VO registry through DaCHS' OAI-PMH endpoint. See the *Operator's Guide* for details.

Arguments

rd RDs to publish; you can give ids or file paths. Use the magic value ALL to check all published RDs, ALL_RECURSE to look for RDs in the file system (check twice for detritus before doing that later thing).

Options

-k, **--keep-timestamps**
Preserve the time stamp of the last record modification. This may sometimes be desirable with minor updates to an RD that don't justify a re- publication to the VO.

-u, **--unpublish**
Unpublish all resources coming from this RD Note that this will in general create "deleted records", i.e., essentially empty resource records only stating that the resource referenced by an identifier is no longer available. This is important for reliable operation in the presence of incremental harvesting.

THE PURGE SUBCOMMAND

Synopsis:

dachs purge [-h] tablename [tablename...]

This will delete tables in the database and also remove their metadata from DaCHS' internal tables (e.g., TAP_SCHEMA, table of published records). Use this if **dachs drop** fails for to

remove some table for one reason or another.

Argument

tablename
(SQL) name of the table to drop, including the schema name

THE SERVE SUBCOMMAND

Synopsis:

dachs serve [-h] {debug | reload | restart | start | stop}

This exposes various functionality for managing DaCHS' server component. While these usually are being called through init scripts or systemd components, the **debug** subfunction is very convenient during service development off the production environment.

serve start, **stop**, and **restart** manage a PID file, which DaCHS does not do when running under systemd. Hence, do *not* use these subcommands when systemd manages your DaCHS. expireRD is fine, though.

Subsubcommands

- **debug** -- run a server and remain in the foreground, dumping all kinds of stuff to the terminal. If you have an SSL certificate, this will try bind an https interface to port 40443; that's exclusively for little experiments, do not use it for actual operations.
- **reload** -- reload server configuration (incomplete)
- **restart** -- restart the server
- **start** -- start the server and put it in the background
- **stop** -- stop a running server
- **expireRD <rdId>** -- forget the cached RD in the running server. While in general, DaCHS reloads an RD if it finds it changed, sometimes it does not realise some dependency has changed. In that case, this subsubcommand can expunge the RD so it will be re-parsed when the next access occurs. Note that this requires [web]adminpasswd be set; the command uses [web]serverURL to figure out where to connect the running server.

The **start** subcommand accepts a **--foreground/-f** option. If you pass it, the server will not double-detach and not redirect stderr to a log file and instead stay in the foreground; it will also not check for PID files or create them. In contrast to **debug**, it will change users, etc., and not put the DaCHS into debug mode. This option is primarily useful for systemd units.

THE START SUBCOMMAND

Synopsis:

dachs start [-h] (list|<data-type-tag>)

The start subcommand generates a template RD for a certain type of data that you can then fill out. The data-type-tag can be something like scs (for catalogs), siap (for images), or ssap (for spectra). Pass **list** to see what is available.

The template uses the name of current directory as resdir and schema name. That means that if starting a data collection, you should create its resdir as a child of GAVO_ROOT/inputs and execute **dachs start** in that dicrectory.

To fill out the template RD, load it into a text editor and, ina first go, search for the pattern %.*%. You should see enough hints from what is between the percent signs and the environment to get a reasonable shot at filling things out. Then reead the comments; very typcially, you can get an extremely basic data publication without that, but a good service will normally require some extra work beyond filling things out.

Argument

data-type-tag

A short identifier for a data type. Pass **list** here to see a list of known tags and their meanings.

THE TEST SUBCOMMAND

Synopsis:

dachs test [-h] [-v] [-V] [-d] [-t TAG] [-R N] [-T SECONDS] [-D FILE]
[-w SECONDS] [-u SERVERURL] [-n NTHREADS]
[--seed RANDOMSEED] [-k KEYWORDS]
id

This runs regression tests embedded in the whatever is reference by id (can be an RD, a regression suite, or a single regression test). For details, see the chapter on *regression testing* in the *DaCHS Reference Manual*.

Argument

id RD id or cross-RD identifier for a testable thing. Write ALL here to have DaCHS search and test all RDs in the inputs, ignoring those in or below directories with a file named DACHS_PRUNE.

Options

-h, **--help**
show this help message and exit

-v, **--verbose**
Talk while working

-d, **--dump-negative**
Dump the content of failing tests to stdout

-t *TAG*, **--tag** *TAG*
Also run tests tagged with TAG.

-R *N*, **--n-repeat** *N*
Run each test N times

-T *SECONDS*, **--timeout** *SECONDS*
Abort and fail requests after inactivity of SECONDS

-D *FILE*, **--dump-to** *FILE*
Dump the content of last failing test to FILE

-w *SECONDS*, **--wait** *SECONDS*
Wait SECONDS before executing a request

-u *SERVERURL*, **--serverURL** *SERVERURL*
URL of the DaCHS root at the server to test

-n *NTHREADS*, **--number-par** *NTHREADS*
Number of requests to be run in parallel

-k *KEYWORDS*, **--keywords** *KEYWORDS*
Only run tests with descriptions containing all (whitespace-separated) keywords. Sequential tests will be run in full, nevertheless, if their head test matches.

THE VALIDATE SUBCOMMAND

Synopsis:

dachs validate [-h] [-x] [-v] rd [rd...]

This checks RDs for well-formedness and some aspects of VO-friendliness

Arguments

rd RD path or ID to validate. Write ALL here to have DaCHS search and validate all RDs in your input and validate them, ignoring those in or below directories with a file named DACHS_PRUNE.

Options

-h, **--help**
show this help message and exit

-p, **--pre-publication**
Validate as if all services were IVOA published even if they are not (this may produce spurious errors if unpublished services are in the RD).

-v, **--verbose**
Talk while working

-t, **--run-tests**
Run regression tests embedded in the checked RDs

-T *SECONDS*, **--timeout** *SECONDS*
When running tests, abort and fail requests after inactivity of SECONDS

-c, **--compare-db**
Also make sure that tables that are on disk (somewhat) match the definition in the RD.

THE UPGRADE SUBCOMMAND

Synopsis:

dachs upgrade

Each DaCHS version has an associated database schema version, encoding the structure of

DaCHS' (and the implemented protocol versions') ideas of how system and user tables should look like. **dachs upgrade** attempts to work out how to change the database to match the expectations of the current version and executes the respective code. It will not touch its data structures if it decrees that the installation is up to date.

Operating system packages will usually try to run **dachs upgrade** as part of their management operation. In case **dachs upgrade** requires manual intervention, this may fail, in which case operators may need to call **dachs upgrade** manually.

Operators keeping a manually installed DaCHS should run **dachs upgrade** after each **svn update** or update from tar.

dachs upgrade cannot perform actions requiring superuser privileges, since none of its roles have those. Currently, this is mainly updating postgres extensions DaCHS uses (if you use extra ones, you can configure DaCHS' watch list in [db]managedExtensions). **dachs upgrade -e** will attempt to figure out the instructions necessary to update extensions and write them to stdout. Hence, operators should execute something like **dachs upgrade -e | psql gavo** from a database superuser account after upgrading postgres extensions.

Options

--force-dbversion *FORCEDBVERSION*
assume this as the database's schema version. If you don't develop DaCHS, you almost certainly should stay clear of this flag

-e, **--get-extension-script**
Dump a script to update DaCHS-managed extensions (will print nothing if no extensions need updating). This will return 0 if material was written, 1 otherwise.

-t, **--update-tap-schema**
Update the TAP schema, i.e., re-read all the metadata from the TAP-published tables. This isn't really related to upgrading, but it is generally a good idea to sync on-disk representations with what you may have edited in the RDs.

THE ADQL SUBCOMMAND

Synopsis:

dachs adql query

This subcommand executes ADQL queries locally and writes the resulting VOTable to stdout.

We consider removing it.

INTERNAL OR DEPRECATED SUBCOMMANDS

The subcommands **show**, **stc** are deprecated and not documented here. They may disappear without further notice.

the subcommands **taprun**, **dlrun**, **uwsrun**, **gendoc**, **raise** are used internally and should not be directly used by DaCHS operators.

REPORTING BUGS

To report bugs and request support, please use our support mailing list *http://lists.g-vo.org/cgi-bin/mailman/listinfo/dachs-support*.

SEE ALSO

Comprehensive, if always incomplete documentation on DaCHS is available in several documents available at *http://docs.g-vo.org/DaCHS/* (upstream site with PDF downloads and the formatted reference documentation) and *http://dachs-doc.readthedocs.io/en/latest/index.html* (with facilities for updating the documents).

COPYRIGHT

AUTHOR

Markus Demleitner <gavo@ari.uni-heidelberg.de>