GAVO DaCHS operator's guide

Author: Markus Demleitner
Email: gavo@ari.uni-heidelberg.de
Date: 2016-03-16

Contents

This document details the configuration and operation of the GAVO DaCHS server. For information on installing the software, please refer to the installation guide, to learn how to import data, see the tutorial. For an overview of the available documentation, see DaCHS documentation

Starting and stopping the server

The `gavo serve` subcommand is used to control the server. `gavo serve start` starts the server, changes the user to what is specified in the [web] user config item if it has the privileges to do so (that's "gavo" by default; you will already have created that user if you followed the installation instructions) and detaches from the terminal.

Analoguosly, `gavo serve stop` stops the server. To reload some of the server configuration (e.g., the resource descriptors, the vanity map, and the /etc/gavo.rc and ~/.gavorc files), run `gavo serve reload`. This does not reload database profiles, and not all configuration items are applied (e.g., changes to the bind address and port only take effect after a restart). If you remove a configuration item entriely, their built-in defaults do not get restored on reload either.

Finally, `gavo serve restart` restarts the server. The start, stop, reload, and restart operations generally should be run as root; you can run them as the server user (by default, gavo), too, as long as the server doesn't try to bind to a privileged (lower than 1025).

All this can and should be packed into a startup script or the equivalent entity for the init systme of your choice. Our Debian package provides a System V-style init script; it is available from http://svn.ari.uni-heidelberg.de/svn/debian-package/gavodachs/trunk/debian/gavodachs-server.dachs.init and should be installed to /etc/init.d/dachs (of course, if you installed the Debian package, the system has already done this for you).

For development work or to see what is going on, you can run `gavo serve debug`; this does not detach and does not change users.

Publication

To "publish" a resource – which means include it either on your site's home page or in what you report to the VO registry –, add a `publish` element to a `service` element or a `register` element to data or table elements. Both of these let you specify the sets the resources shall be published to. Unless you have specific application, only two sets are relevant: `ivo_managed` for publishing to the VO (see Registry Matters, and `local` for publishing to your data center's service roster. Other sets can be introduced and used for, e.g., specific sub-rosters.

The `publish` element needs, in addition, a render attribute, giving a comma-separated list of renderers the publication is for. The various renderers are translated into capability element in the VO resource records. For example, a typical pattern could be:

```
<publish render="scs.xml" sets="ivo_managed"/>
<publish render="form" sets="local,ivo_managed"/>
```

This generates a capability each for the simple cone search and a browser-based interface; the browser-based interface is, in addition, listed in the local service roster.

You can also publish tables; for those, the notion of renderers make no sense, so the publish element doesn't have that. Instead, you could define services that serve that data. For many cases, you don't even need to do this, since for tables that have `adql="True"`, the local TAP service is automatically considered to be a service for that data.

So, to publish an ADQL-queriable table to the VO for querying via TAP, just write:

```
<publish/>
```

within the table element. A table containing, e.g., data that's queried in a SIAP service in a different RD, would require something like:

```
<publish service="other/rd#siapsvc/>
```

Once you have done this, run `gavo pub <rdid>`. This causes all publishable items in the RD to be published. It also unpublishes everything that was published through the RD before and is no longer published. If the `serverURL` config item on the machine running gavo pub is pointing to the actual running server, the latter will automatically be made aware of these changes. Otherwise, you need to prod the server as discussed in Managing Runtime Resources.

Registry Matters

As explained in the tutorial chapter on the registry, you must provide enough data to allow the VO to tell who you are before the VO registry can usefully include your registry records.

Once that's done, you can add the publishing registry included in DaCHS to the list of registries harvested by VO full registries.

This chapter tries to guide you through this process.

Defining Basic Metadata

The first step is to define your authority (i.e., something like org.g-vo.dc) in your config (`/etc/gavo.rc`), in the `[ivoa]authority` item. Please make sure that the authority is not already taken by someone else; you have probably fulfilled your due diligence if you've run an authority query against the registry and did not find a match. Using your DNS name is not a bad idea. Please don't repeat our (the GAVO DC's) mistake and invert the sequence of particles in your DNS name. Also, this is just the name, there is no ivo:// or other decoration.

Then, add metadata about yourself in `GAVO_ROOT/etc/defaultmeta.txt`. It is a file in the meta stream format; basically it's lines of `<key>: <value>`.

In defaultmeta.txt, you must give basic information on your authority and some fallback for services (the defaults all make no sense at all and should never escape to the VO registry):

- publisher – A short, human-readable name for you
- publisherID – An IVOA id for yourself; set this to ivo://<authority>/org unless you know what you are doing
- contact.name – A human-readable name for some entity people should write to. This is not necessarily different from publisher, but ideally people can write "Dear <contact.name>" in their mails.
- contact.address – A contact address for surface mail
- contact.email – An email address. It will be published on web pages, so there probably should be some kind of spam filter in front of it.
- contact.telephone – A telephone number people can call if things really look bad.
- creator.name – A name to use when you give no creator in your resource descriptors. Could be some error sentinel ("we foget to give credit, please complain") or just contact.name if you produce resources yourself.

- creator.logo – A URL for a logo to use when none is given in the resource metadata. Use a small PNG here.
- site.description – A description of your site (i.e., "data center") Example: `The GAVO data center provides VO publication services to all interested parties on behalf of the German Astrophysical Virtual Observatory.` (use backslashes an the end of the lines to break long lines).

Then, fill out the metadata for the system registry resources in your userconfig RD. This is the stuff in the `registry-interfacerecords` stream (which you can copy from //userconfig if it's not yet in your `etc/userconfig.rd`. Fill things out; in particular everything where there's a `\\metaString` expansion. This still is filled from defaultmeta.txt, but as we want to get rid of that file on the long run, just enter the text as you see fit.

In authority, change in particular

- creationDate – A UTC datetime (with trailing Z); technically, it should be the date the resource record is created, but realistically, just use "now" at the time your're writing the defaultmeta.txt. Example: `2007-12-19T12:00:00Z`.
- title – A human-readable descriptor of what the authority corresponds to. Example: `The Utopia Observatory Data Center`
- description – A sentence or two on what the authority you are using means. This could be the same as site.description if all you're claiming authority for is that; if you're claiming authority for your institute or organisation, this obviously should be different. Example: `The Data Center at the Observatory of Utopia manages lots of substantial data sets created by the bright scientists all over the world` (use backslashes an the end of the lines to break long lines).
- shortName – a short (about 16 chars) identifier for your authority. Example: GAVO DC.
- referenceURL – A URL at which people can learn more about your data center. Example: `http://www.g-vo.org`.
- managingOrg – an ivo id of the organisation you're running the dc for. The default is ivo://<your authority>/org, the content defined in the `manager` resRec. If your institution has a registry entry independent of your DC, you can enter that IVORN here as well (and you would remove the `manager` resRec).

In the `manager` resRec (if you have it), change:

- creationDate – as above for authority.
- title – the name of the organisation on behalf of which you are running the data center. Example: `Observatory of Utopia`
- description and referenceURL – as the analoguous item for authority, just for the organisation for which you are running the data center (e.g., your "home institute"). Example: `The Observatory of Utopia is Lilliput's largest astronomical institution with ten large telescopes spread around the Plain Mountains. Beautiful vistas and lush valleys make them an attractive holiday spot as well. Book now at the Observatories soft money department at 1-800-GOOD-GREED.`

After you've specificed all that, you're ready to define your first resources, viz, your registry itself, the authority, and the organisation that's managing it. These are predefined using the data you just filled in in the //services RD. To publish them, you say:

```
gavo pub //services
```

Registering DaCHS-external Services

The registry interface of DaCHS can be used to register entities external to DaCHS; actually, you're already doing this when you're claiming an authority.

To register a non-service "resource", you can fill out a resRec RD element. You could reserve an RD (say, `GAVOROOT/inputs/ext.rd` to collect such external registrations, or you could put them alongside internal services into their respective RDs. You will then usually just use the resRec's id attribute to determine the IVORN of resource record. It will then be `ivo://<your authority>/<rd id>/<id of resRec>`.

In all likelihood, however, you will want to register services. To do that, use a normal service definition with with a nullCore. You probably need to manually give an accessURL. The most common case is that of a service with a `WebBrowser` capability. These result from `external` or `static` renderers. Thus, the pattern here usually is:

```
<service id="myservice" allowed="external">
  <nullCore/>
  <meta>
    shortName: My external service
    description: This service does wonderful things, even though\
      it's not based on GAVO's DaCHS software.
  </meta>
```

```
  <publish render="external" sets="ivo_managed">
    <meta name="accessURL">http://wherever.else/svc</meta>
  </publish>
</service>
```

Of course, you will normally need to add further metadata as discussed above. `gavo pub` should complain if there's metadata missing, though.

The "services" can be fairly funky, actually; here's how GAVO registers their ADQL reference card:

```
<service id="adqlref" allowed="external">
  <nullCore/>
  <meta>
    shortName: GAVO ADQL ref
    creationDate: 2012-11-05T14:24:00Z
    title: The GAVO ADQL reference card
    subject:Virtual Observatory
    subject:Standards
    subject:ADQL
    description: GAVO's ADQL reference card briefly gives an overview \
    of the SQL dialect used in the VO.  It is available as a PDF\
    file and as Scribus source under the CC-BY license.
    referenceURL:http://www.g-vo.org/pmwiki/About/ADQLReference
  </meta>
  <publish render="external" sets="ivo_managed,local">
    <meta name="accessURL">http://docs.g-vo.org/adqlref/adqlref.pdf</meta>
  </publish>
</service>
```

It is likely that if you register external services, you'll want to manage authorities other than `[ivoa]authority` as used by DaCHS. If you do, just add authority record(s) as before in the `registry-interfacerecords` STREAM in your userconfig RD. And do not forget to add lines like:

```
<meta name="managedAuthority">edu.euro-vo.org</meta>
```

within the `<service id="registry"` in the user config.

Registering Web Interfaces (And More)

A typical situation is that you have a standard service (SSA, SCS, SIAP, etc) and a form-based custom service on the same data. Since the form-based service caters to humans, it can require quite different input parameters (and thus usually cores) and output tables, and so you'll usually have a different service on it.

If you want to publish both services to the VO, you could add `publish` elements with `sets="ivo_managed"` to both `service` elements – but that would yield two resource records (which you then should link via relatedTo metas). At least when the form interface doesn't add significant functionality, this would usually seem overkill – e.g., your service would show up twice in resource listings.

Therefore, it is typically preferable to add the web interface as a capability to the the resource record of the standard service. To let you do that, the `publish` element takes an optional `service` attribute containing the id of a service that should be used to fill the capability's metadata.

Here's an example:

```
<service id="web" defaultRenderer="form">
  <meta name="title">Form-based service</meta>
  <!-- add this service to the local roster -->
  <publish render="form" sets="local"/>
  ...
</service>

<service id="ssa" allowed="form,ssap.xml">
  <publish render="ssap.xml" sets="ivo_managed"/>
  <!-- now make a WebBrowser capability on this service in the IVOA
  published resource record, based on the service with the id web -->
  <publish render="form" sets="ivo_managed" service="web"/>
  ...
</service>
```

To publish

Simple OAI operation

If you want to check what you have published, see the `/oai.xml` on your server, e.g., http://localhost:8080/oai.xml. This is a plain OAI-PMH interface with some style sheets (if you want to customize them, copy them to `rootDir/web/xsl/`). The default style sheets add a link to "All identifiers defined here". Follow it to a list of all records you currently publish.

The OAI endpoint can also be used to help you in debugging validity problems with your registry content. To XSD-validate your registry without bothering the RofR (see above), you can do the following:

```
curl <your oai.xml url>?verb=ListRecords&metadataPrefix=ivo_vor |\
  xmlstarlet fo > toval.xml
gavo admin xsdValidate toval.xml
```

This may result in a few error messages; if you don't understand them, it's a good idea to just go to the respective line in toval.xml and give it a long, hard look.

Making the VO see your Registry

The VO registry is a distributed system. There still is some sort of root, the Registry of Registries or RofR. Once your system provides sufficient metadata, go to http://rofr.ivoa.net/regvalidate/regvalidate.html and enter your registry endpoint (i.e., your installation's root URL with /oai.xml appended).

GAVO DaCHS is lenient with missing metadata and will deliver invalid VOResource for records missing some. It is not unlikely that your registry will not validate on the first attempt. Reading the error messages should give you a hint what's wrong. You can also use the `gavo val` command on the RDs that generate invalid records to figure out what's wrong.

Once your registry passes the validation test, you can add it to the RofR, and the full registries will start to harvest your registry (after a while).

Adapting DaCHS for Your Site

As delivered, the web interface of DaCHS will make it seem you're running a copy of the GAVO data center, with some metadata defused such that you are not actually disturbing our operation if you accidentally activate your registry interface. You should thus first customize the items given in `etc/defaultmeta.txt` (as discussed in Registry Matters).

The next adaptations are done through the configuration (as discussed in Configuration Settings, i.e., usually in /etc/gavo.rc). The most relevant item here is `[web]sitename`, which should contain a terse identifier for the site (like "GAVO Data Center"). It is shown in titles and top headlines in many places. If you plan to use DaCHS' embargo feature together with user authorisation, you must also set `[web]realm` to some characteristic string. You could use the site name here; some user agents use it to display a prompt like "Credentials for <realm>" or similar.

If you want, you can set `[web]favicon` to either a webDir-relative path or a full URL to a favicon.

It is also advisable to configure `[general]maintainerAddress` to a mail address of a person who will read problem reports. DaCHS doesn't send many of those yet, but it's still valuable if the software can cry for help if necessary. Sending mail only works if the local machine can actually send mail. If there is no MTA

on your machine yet, we recommend `nullmailer` as a lightweight and easy-to-configure sendmail stand-in. If you use something else, you may need to adapt `[general]sendmail`.

For the rest, you can customize almost everything by overriding built-in resources. There are five major entities that you can override:

- customisation hooks
- userconfig RD
- Simple Web Resources
- Templates
- Overridden System RDs

If you find you need to override anything but the logo, please talk to us first – we'd in general prefer to provide customisation hooks. Overridden distribution files are always a liability on upgrades.

Customisation Hooks

Operator CSS

To override css rules we distribute or add new rules, avoid changing gavo_dc.css as described in Simple Web Resources, as that will be a liability when upgrading. Instead, drop a CSS file somewhere (recommended location: GAVO_ROOT/web/nv_static/user.css) and add a configuration item in `[web]operatorCSS`. With the recommended location, this would work out to be:

```
[web]
operatorCSS: /static/user.css
```

in /etc/gavo.rc.

This can also be an external URL, but we recommend against that, as that would force a browser to open one external connection per web page delivered.

By far the most common complaint is that we are limiting the width of p and li elements to 40em. We believe that text lines longer than about 80 characters are hard to read and should be avoided. On pages with tables where users might actually want to run browsers filling the entire screen, this choice cannot be made through a sensible choice of the width of the user agent window on the user side but requires CSS intervention.

Having said that, if you really think you want window-filling text lines, just put:

```
p, li {
        max-width: none;
}
```

into your operator CSS.

XSL configuration

DaCHS employs client-side XSLT for some purposes -- for instance, to show OAI-PMH (registry) responses in web browsers, to allow perusing datalink results in the browser, and to allow web browsers some rudimentary interaction with UWS applications like TAP.

The default XSLT contains references to the GAVO data center; to change these (or something else), override the xsl config stylesheet, which is expected at /static/xsl/dachs-xsl-config.xsl. The recommended way to go about this is:

```
cd /var/gavo  # or whereever your DaCHS root is
cd web/nv_static
mkdir -p xsl
cd xsl
gavo admin dumpDF web/xsl/dachs-xsl-config.xsl > dachs-xsl-config.xsl
```

Then edit `dachs-xsl-config.xsl`. Note that you have to restart the server once to make it notice the override.

Userconfig RD

Fairly new in DaCHS is an RD exclusively for configuration. This is a place in which you can put streams that fill certain hooks; we expect to move more configuration into userconfig.

DaCHS has a builtin RD `//userconfig` that is updated as you update DaCHS. It always contains fallbacks for everything that can be in userconfig used by the core code. To override something, pull the elements in questions in your own userconfig RD and edit it there.

Your own userconfig RD is expected in `$GAVO_DIR/etc/userconfig.rd`. If it's not there yet, there's nothing wrong with starting with the distributed one:

```
cd ‘gavo config configDir‘
gavo admin dumpDF //userconfig > userconfig.rd
```

Once it's already there, use `dumpDF //userconfig` and, say, `less` to pick out the templates for whatever elements you need to copy. Currently, userconfig is already used in configuring the registry interface and extending the built-in obscore schema.

Changes to userconfig.rd are picked up by DaCHS but will usually not be visible in the RDs they end up in. This is because DaCHS does not track which RDs make use of userconfig, so these will typically need to be reloaded manually. For instance, if you changed TAP examples, you'd need to run:

```
gavo serve exp //tap
```

to make your change show up in the web interface. Although usually not necessary, you can reload userconfig itself using:

```
gavo serve exp %
```

Simple Web Resources

For items coming from `static` (e.g., images, css, javascript), this overriding works by dropping same-named files in `$GAVO_ROOT/web/nv_static`.

Thus, you should put a PNG of your logo into `$GAVO_ROOT/web/nv_static/img/logo_medium.png`.

Other files you may want to override in this way include

- `css/gavo_dc.css` – the central CSS; you could use this for skinning. However, we recommend you just add an `@import url("<your css url>");` to the file the server delivers by default, since some of the css is almost necessary, and you want easy upgrade paths when we change the master CSS.
- `help.shtml` – the help file. Unfortunately, we blurb quite a lot about GAVO in there right now. We'll think of something more parametrisable, but meanwhile you may want to have your own version
- `img/logo_big.png`, `img/logo_tiny.png` – scaled versions of your logo; logo_big should be 200 pixels wide or more, logo_tiny of order 50 pixels wide.
- `js/gavo.js` – could be the place for additional javascript; but frankly, if you want custom javascript, write to us and we'll think of a sane mechanism.

- `xsl/oai.xsl`, `xsl/uws-joblist-to-html.xsl`, `xsl/uws-job-to-html.xsl`, and `vosi.xsl` – XSLT stylesheet files. If you override these to customize them, please let us know. We'd try to put out generic stylesheets that are customisable without having to muck around in stuff that's basically functionality.

Templates

There is now a document on HTML templating in DaCHS

Overridden System RDs

You can copy system RDs from `gavo/resources/inputs/__system__` in the distribution to `$GAVO_ROOT/inputs/__system__` (adapt if you have played tricks with `inputsDir`) and edit them there. Again, if you feel you need to do that, contact us first, maybe we can work something out; it's a liability for upgrades.

Other documents

The default help file and the default sidebar link to a privacy policy that you should put down in `$GAVO_ROOT/web/nv_static/doc/privpol.shtml`. The document must be well-formed XHTML. Also, files with an extension shtml will be interpreted as templates over the service `//services/root`, which means that you can use the usual render functions and data items; the same goes for `disclaimer.html` (referenced from the standard sidebar) and, if you offer SOAP services, `soaplocal.html`. See the respective pages in the GAVO DC (`http://dc.g-ov.org/static/doc/...`) for ideas as to what to include.

The Vanity Map

DaCHS' URL scheme leads to somewhat clunky URLs that, in particular, reflect the file system underneath. While this doesn't matter to the VO registry, it is possibly unwelcome when publishing URLs outside of the VO. To overcome it, you can define "vanity names", single path elements that are mapped to paths.

These mappings are read from the file `GAVO_ROOT/etc/vanitymap.txt`. The file contains lines of the format:

```
<target> <key> [<option>]
```

Target is a path that must *not* include nevowRoot and must *not* start with a slash (unless you're going for very special effects).

Key normally is a single path element (i.e., a string without a slash). If this path element is found in the first segment, it is replaced with the segments in target.

<option> can only be `!redirect` or empty right now.

If it is `!redirect`, `<key>` may be a path fragment (as opposed to a single path element); leading and trailing slashes are ignored. If the *enire* query path matches this key, a redirect to this key is generated. This is intended to let you shut down services and introduce replacements. If the incoming URL contains a query, it will be appended to the replacement URL. Thus, even stored queries or forms can potentially work across such a redirect.

You can also (ab)use the redirect option to give vanity names, but since the target will show up in the browser address line, normal maps are highly preferred. The only time normal maps don't work for this is when the resource directory is identical to the vanity name (you'll get an endless loop then), so you should avoid that situation.

Empty lines and #-on-a-line-comments are allowed in the input.

As an example, here's the vanity map that DaCHS had builtin as of version 0.6:

```
__system__/products/p/get getproduct
__system__/services/registry/pubreg.xml oai.xml
__system__/services/overview/external odoc
__system__/dc_tables/show/tablenote tablenote
__system__/dc_tables/show/tableinfo tableinfo
__system__/services/overview/admin seffe
__system__/tap/run/tap tap
__system__/adql/query/form adql !redirect
```

Note again that `<key>` must be a *single* path element only.

Configuration Settings

Many aspects of the data center can be configured using INI-style configuration files. DaCHS tries to obtain them from a global location (`/etc/gavo.rc` or whatever is in the `GAVOSETTINGS` environment variable) and a user-specific file (`~/gavo.rc` or whatever is in the `GAVOCUSTOM` variable). The server should probably be configured in the global location exclusively, since otherwise it will behave differently depending on which user starts the server.

This section starts with a walkthrough through the more relevant settings, section by section; below, there is a reference of all supported configuration items.

Walkthrough

The general Section

This mainly sets paths. The most important is `rootDir`, a directory most other paths are relative to. This is the one you'll most likely want to change. If you, e.g., wanted to have a private DaCHS tree, you could put:

```
[general]
rootDir: /home/user/gavo
```

into the personal configuration file (which DaCHS searches in `~/.gavorc`) by default; this would then override the analogous specification in `/etc/gavorc`.

The other paths in this section are interpreted relative to rootDir unless they start with a slash.

You may want to set tempDir and cacheDir to a directory local to your machine if rootDir is mounted via a network. Also note that we do no synchronisation for writing to the log (and never will -- we will provide syslog based logging if necessary), so you may want to tweak logDir too to keep actions from seperate users seperate.

The web Section

You typically want to adapt several settings here. First `bindAddress` gives the IP address of the interface DaCHS will accept requests from. By default, that's localhost, meaning that your server will only talk to the machine it runs on. Once you want to serve other people, you will need to change this. For most systems, binding to all interfaces is what you want; keep bindAddress empty to accomplish that.

You may also want to change `serverPort`. That is the TCP port DaCHS listens to. The default, 8080, is what's commonly used in test setups. On machines dedicated to DaCHS, you would set it to 80, the standard HTTP port; this will of course fail if there's already another web server running.

DaCHS frequently needs to produce full URLs to itself. To do that, it uses `serverURL`. While we could potentially infer that from `bindAddress` and `serverPort`, today's web setups are frequently too complicated to make that work. So, adapt `serverURL`, too, to the base URL of your server, without any trailing slash. A complete setup for a public server would thus look like this:

```
[web]
bindAddress:
serverPort: 80
serverURL: http://mydc.myvo.org
```

Note that `serverURL` *must* include the port if it is not 80 (or 443 for https). If you actually kept the default and just put the machine on the net, your web section would need to include something like:

```
[web]
bindAddress:
serverURL: http://your.machine.example.org:8080
```

– the empty `bindAddress` is necessary so DaCHS doesn't just bind to the loopback address, the `serverURL` because DaCHS has no way of knowing the preferred name of the machine it's running under; it *could* add the port, which it knows, but doing that would, e.g., make the lives of people operating behind reverse proxies hard.

While you are at it, set `sitename` to a short string describing your server (this is currently only used in the registry interface).

You will probably also want to set `adminpasswd`. If set, you can log in on your server as user gavoadmin with this password. Gavoadmin basically may do everything (access protected resources, clear caches, etc). The password is given in clear text; doing some kind of encryption would only make sense if you were prepared to enter some kind of passphrase every time you start the server. As in other places, DaCHS assumes the machine it runs on is trusted.

The db Section

In the db section, some global properties of the database access layer are defined. Currently, the most releveant one is profilePath. This is a colon-separated list of rootDir-relative paths in which DaCHS looks for database profiles (expansion of home directories is supported). The first match in any of these directories wins. This is useful when you have a test setup and a production setup -- just say `include dsn` in the common profiles (by default in configDir) and have separate dsn files in the ~/.gavo directories of the accounts feeding the test and production databases.

You probably do not want to to mess with any settings ending in Roles. These are for rather exotic setups where DaCHS needs to accomodate other software.

The profiles Section

The profile section maps profile names to file names. These file names are relative to any of the directories in db.profilePath. Usually, you should keep whatever gavo init has come up with and hence not change anything here.

The profiles contain a specification of the access to the database in (unfortunately yet another, but simple) language. Each line in such a profile is either a comment (starting with #), an assignment (with "=") or an instruction (consisting of a command and arguments, separated by whitespace).

Keywords available for assignment are

- host -- the host the database resides on. Leave empty for a Unix socket connection.
- port -- the port the database listens on. Leave empty for default 5432.
- database -- the database your tables live in.
- user -- the user through which the db is accessed.
- password -- the password of user.

There's just one command available, viz.,

- include -- read assignments and instructions from the profile given in the argument

`gavo init` creates four profile files, `dsn`, `feed`, `trustedquery`, and `untrustedquery`. These are referred to in the default profiles section, and are basically required by the python code.

Reference

You can get an up-to-date version of this by running `gavo config`.

Section [general]

Paths and other general settings.

- cacheDir: path relative to rootDir; defaults to 'cache' -- Path to the DC's persistent scratch space

- configDir: path relative to rootDir; defaults to 'etc' -- Path to the DC's non-ini configuration (e.g., DB profiles)
- defaultProfileName: string; defaults to '' -- Deprecated and ignored.
- gavoGroup: string; defaults to 'gavo' -- Name of the unix group that administers the DC
- group: string; defaults to 'gavo' -- Name of the group that may write into the log directory
- inputsDir: path relative to rootDir; defaults to 'inputs' -- Path to the DC's data holdings
- logDir: path relative to rootDir; defaults to 'logs' -- Path to the DC's logs (should be local)
- logLevel: value from the list info, debug, warning, error; defaults to 'info' -- How much should be logged?
- maintainerAddress: string; defaults to '' -- An e-mail address to send reports and warnings to; this could be the same as contact.email; in practice, it is shown in more technical circumstances, so it's adviable to have a narrower distribution here.
- operator: string; defaults to '' -- Deprecated and ignored. Use contact.email in defaultmeta.txt instead.
- platform: string; defaults to '' -- Platform string (can be empty if inputsDir is only accessed by identical machines)
- rootDir: string; defaults to '/var/gavo' -- Path to the root of the DC file (all other paths may be relative to this
- sendmail: string; defaults to 'sendmail -t' -- Command that reads a mail from stdin, taking therecipient address from the mail header, and transfers the mail (this is for sending mails to the administrator). This command is processed by a shell (generally running as the server user), so you can do tricks if necessary.
- stateDir: path relative to rootDir; defaults to 'state' -- Path to the DC's state information (last imported,...)
- tempDir: path relative to rootDir; defaults to 'tmp' -- Path to the DC's scratch space (should be local)
- uwsWD: path relative to rootDir; defaults to 'state/uwsjobs' -- Directory to keep uws jobs in. This may need lots of space if your users do large queries

- webDir: path relative to rootDir; defaults to 'web' -- Path to the DC's web related data (docs, css, js, templates...)
- xsdclasspath: shell-type path; defaults to 'None' -- Classpath necessary to validate XSD using an xsdval java class. You want GAVO's VO schemata collection for this. Deprecated, we're now using libxml2 for validation.

Section [adql]

Settings concerning the built-in ADQL core

- webDefaultLimit: integer; defaults to '2000' -- Default match limit for ADQL queries via a web form

Section [async]

Settings concerning TAP, UWS, and friends

- csvDialect: string; defaults to 'excel' -- CSV dialect as defined by the python csv module used when writing CSV files.
- defaultExecTime: integer; defaults to '3600' -- Default timeout for UWS jobs, in seconds
- defaultExecTimeSync: integer; defaults to '60' -- Default timeout for synchronous UWS jobs, in seconds
- defaultLifetime: integer; defaults to '172800' -- Default time to destruction for UWS jobs, in seconds
- defaultMAXREC: integer; defaults to '2000' -- Default match limit for ADQL queries via the UWS/TAP
- hardMAXREC: integer; defaults to '20000000' -- Hard match limit (i.e., users cannot raise MAXREC or TOP beyond that) for ADQL queries via the UWS/TAP
- maxTAPRunning: integer; defaults to '2' -- Maximum number of TAP jobs running at a time

Section [db]

Settings concerning database access.

- adqlProfiles: set of strings; defaults to 'untrustedquery' -- Name(s) of profiles that get access to tables opened for ADQL
- defaultLimit: integer; defaults to '100' -- Default match limit for DB queries
- interface: string; defaults to 'psycopg2' -- Don't change
- maintainers: set of strings; defaults to 'admin' -- Name(s) of profiles that should have full access to gavo imp-created tables by default
- msgEncoding: string; defaults to 'utf-8' -- Encoding of the messages coming from the database
- profilePath: shell-type path; defaults to '~/.gavo:$configDir' -- Path for locating DB profiles
- queryProfiles: set of strings; defaults to 'trustedquery' -- Name(s) of profiles that should be able to read gavo imp-created tables by default

Section [ivoa]

The interface to the Greater VO.

- authority: string; defaults to 'x-unregistred' -- The authority id for this DC; this has *no* leading ivo://
- dalDefaultLimit: integer; defaults to '10000' -- Default match limit on SCS/SSAP/SIAP queries
- dalHardLimit: integer; defaults to '1000000' -- Hard match limit on SCS/SSAP/SIAP queries (be careful: due to the way these protocols work, the results cannot be streamed, and the results have to be kept in memory; 1e7 rows requiring 1k of memory each add up to 10 Gigs...)
- oaipmhPageSize: integer; defaults to '500' -- Default number of records per page in the OAI-PMH interface
- sdmVersion: value from the list 1, 2; defaults to '1' -- Version of the spectral data model we generate our spectra as (unless someone asks for another version explicitly).
- votDefaultEncoding: value from the list binary, td; defaults to 'binary' -- Default 'encoding' for VOTables in many places (like the DAL responses; this can be user-overridden using the _TDENC local HTTP parameter.

Magic Section [profiles]

Ignored and deprecated, only here for backward compatibility. The items in this section are all of type profile name. You can add keys as required.

Section [ui]

Settings concerning the local user interface

- outputEncoding: string; defaults to 'iso-8859-1' -- Encoding for system messages. This should match what your terminal emulator is set to

Section [web]

Settings related to serving content to the web.

- adminpasswd: string; defaults to '' -- Password for online administration, leave empty to disable
- adsMirror: string; defaults to 'http://ads.g-vo.org' -- Root URL of ADS mirror to be used (without a trailing slash)
- bindAddress: string; defaults to '127.0.0.1' -- Interface to bind to
- enableTests: boolean; defaults to 'False' -- Enable test pages (don't if you don't know why)
- favicon: path relative to webDir; defaults to 'None' -- Webdir-relative path to a favicon
- graphicMimes: list of strings; defaults to 'image/fits,image/jpeg' -- MIME types considered as graphics (for SIAP, mostly)
- jsSource: boolean; defaults to 'False' -- If True, Javascript will not be minified on delivery (this is for debugging)
- maxPreviewWidth: integer; defaults to '300' -- Ignored, only present for backward compatiblity
- maxUploadSize: integer; defaults to '20000000' -- Maximal size of file uploads in bytes.
- nevowRoot: path fragment; defaults to '/' -- Path fragment to the server's root for operation off the server's root; this must end with a slash (and, frankly, if you must use this feature, you'll probably encounter some bugs. we want to fix those, though.)

- preloadRDs: list of strings; defaults to '' -- RD ids to preload at the server start (this is mainly for RDs that have execute children that should run regularly).
- previewCache: path relative to webDir; defaults to 'previewcache' -- Webdir-relative directory to store cached previews in
- realm: string; defaults to 'X-Unconfigured' -- Authentication realm to be used (currently, only one, server-wide, is supported)
- serverPort: integer; defaults to '8080' -- Port to bind the server to
- serverURL: string; defaults to 'http://localhost:8080' -- URL fragment used to qualify relative URLs where necessary. Note that this must contain the port the server is accessible under from the outside if that is not 80.
- sitename: string; defaults to 'Unnamed data center' -- A short name for your site
- sqlTimeout: integer; defaults to '15' -- Default timeout for db queries via the web
- templateDir: path relative to webDir; defaults to 'templates' -- webDir-relative location of global nevow templates
- user: string; defaults to 'gavo' -- Run server as this user.
- voplotCodeBase: URL fragment relative to the server's root; defaults to 'None' -- Deprecated and ignored.
- voplotUserman: URL fragment relative to the server's root; defaults to 'Deprecated and ignored' -- URL to the documentation of VOPlot

Managing Runtime Resources

DaCHS caches quite a lot of information rather aggressively, which means that editing information on disk may not immediately influence the behaviour of the server. This is particularly true for the default meta (`etc/defaultmeta.txt`), the vanity name translations (`etc/vanitynames.txt`), and the database profiles. Most of this can is reloaded on `gavo serve reload`, but certain settings (like `serverPort` and `bindAddress`) only take effect on a restart.

The resource descriptors are special. The server should pick up edits on RDs automatically, with the following exceptions:

- Built-in `__system__` RDs are not controlled. The main reason here is that these may not actually have disk files behind them, depending on the installation they may come from an archive.

- Only the file the RD was loaded from is checked. This means that if you override a built-in `__system__` RD with your own version in `inputs/__system__`, DaCHS will not automatically pick that up
- If you access an RD with no corresponding file and create that file afterwards, that change will also not be picked up automatically.

`gavo serve reload` will reload even those RDs. To selectively invalidate RDs that fall under these categories when you don't want to reload or restart the server, use the administration panel for the RD through the webserver; see Admin Web Interfaces

Admin Interfaces

Admin Web Interfaces

Some operation on the data center can be done from its web interface. To use these features, you have to set the `[web] adminpasswd` configuration item. You can then use the "Log in" link in the side bar using `gavoadmin` as the user name.

If you are logged in as gavoadmin, you should see an "Admin me"-link in the side bar of services. The page behind that link lets you block all services on the respective RD – where blocking means all requests are rejected until the RD is reloaded – and reload the RD. This is the recommended way to notify DaCHS that an RD has changed and needs re-reading.

In the form, you can also set scheduled down times. This is for VOSI, an interface clients could use to figure out whether a service can reasonable be expected to work. Since there don't seem to be clients exploiting the VOSI endpoints for such purposes so far, you probably don't need to bother.

You can directly access the administraion panel for an RD by accessing `/seffe<rdId>`, e.g , `seffe/__system__/services`.

There are several more or less introspective resources within DaCHS that do not need authentication.

Among those, there's `/browse`. That's a list of all RDs that have (ivo or local) published services or data in them. Links on the RDs lead to info pages on the RDs, in particular giving tables and services within the RD.

robots.txt

DaCHS answers to requests for robots.txt with a built-in resource that forbids to index URLs with `/seffe` and `/login`. You may want to keep other pages out

of indices. In particular, `/browse` will let robots find unpublished services. To exclude those, add a file `robots.txt` in your `webDir` (run `gavo config webDir` to find out where that is) and add lines like:

```
Disallow: /browse
```

The built-in rules will be prepended to whatever you specify in your user robots.txt. For more information on what you can put into robots.txt, see Robot exclusion standard

Admin CLI Interfaces

You can also perform various housekeeping operations using `gavo admin`. Try `gavo admin --help`. This includes user management (there's a bit on it in the tutorial), precomputing previews for images, and create registry records for deleted services that got lost.

An admin tool that comes in handy is `gavo admin tapabort`. Call it with a TAP job id and a helpful (!) text to abort a TAP job and set it to an error state giving a short explanation what happened and the helpful text. The idea is that when users run queries against large tables without using indices (or do other stupid things), you can send them messages in this way (and clear away their resource hogs at the same time).

Upgrading

Upgrading DaCHS

In general, we try to make upgrades painless, but with a system allowing people to play tricks with intenstines like DaCHS guarantees are hard. Be sure to subscribe to DaCHS-users. We'll announce new releases there, together with brief release notes pointing to possible spots of trouble. Ideally, you'll have a development system and regression tests in place that let you diagnose problems before going to production. Poke us for hints on good and easily-maitained setups.

Upgrading Installations from Debian Package

That's easy:

```
apt-get update
apt-get upgrade
```

After the upgrade, be sure to run your regression tests (if you have defined any):

```
gavo test ALL
gavo val -c ALL
```

This last command might complain about mismatches between RD and on-disk metadata; there are several reasons why that may happen, including dumb or clever things we've done in the software (in the latter case, the Changes file should tell you about it). In any case, you should fix the problem, usually by re-importing the respective table.

Upgrading Installations from SVN

The basic thing to remember: After every update, do, as a user with ingestion privileges, the restart sequence:

```
$ gavo val -c ALL
$ sudo /etc/init.d/dachs stop # (or whatever you use to stop the server)
$ gavo upgrade
$ sudo /etc/init.d/dachs start # (or whatever you use to start the server)
$ gavo test ALL
```

In principle, gavo upgrade can run while the server is active, and with most updates, users won't even see errors, but since you need to restart anyway, why bother. On possible failures of the gavo val command, see the last paragraph of the previous section.

The steps to update depend on what you did to install out of the subversion checkout. If you initially said `setup.py develop` (which we recommend), all it takes to upgrade is:

```
$ cd <checkout dir>
$ svn update
<run the restart sequence given above>
```

If you instead initially said `setup.py install`, do:

```
$ cd <checkout dir>
$ svn update
$ sudo python setup.py install
<run the restart sequence given above>
```

Upgrading Postgres

There's a howto over at http://docs.g-vo.org/DaCHS/howDoI.html#upgrade-the-database-engine