Common Problems with DaCHS

Author: Markus Demleitner
Email: gavo@ari.uni-heidelberg.de
Date: 2018-10-02

Contents

This document tries to discuss some error messages you might encounter while running DaCHS. The rough idea is that when you can grep in this file and get some deeper insight as to what happened and how to fix it.

We freely admit that some error messages DaCHS spits out are not too helpful. Therefore, you're welcome to complain to the authors whenever you don't understand something DaCHS said. Of course, we're grateful if you checked this file first.

DistributionNotFound

When trying to run any program, you may see tracebacks like:

```
Traceback (most recent call last):
  File "/usr/local/bin/gavo", line 5, in <module>
    from pkg_resources import load_entry_point
  [...]
  File "/usr/lib/python2.6/dist-packages/pkg_resources.py", line 552, in resolve
    raise DistributionNotFound(req)
pkg_resources.DistributionNotFound: gavodachs==0.6.3
```

This is usually due to updates to the source code when you have installed your source in development mode. Simple do `sudo python setup.py develop` in the root of the source distribution again.

Another source of this error can be unreadable source directories. Please check that the user that's trying to execute the command can actually read the sources you checked out.

'gavodachs' package upgrade fails

If you try to upgrade an older version (< 0.9) of the 'gavodachs' package, e.g. by typing:

```
sudo apt-get update && sudo apt-get upgrade
```

it could happen that you run into troubles when the gavodachs server is going to be stopped (and restarted). If the server stop fails, the installation of the 'gavodachs-server' package will aborted which leaves the package in a half-configured state. The corresponding error message would be something similar to:

```
Stopping VO server: dachsTraceback (most recent call last):
  [...]
  File "/usr/lib/python2.7/dist-packages/pkg_resources.py", line 584,
    in resolve raise DistributionNotFound(req)
pkg_resources.DistributionNotFound: gavodachs==0.9
```

In that case try:

```
sudo dpkg --remove --force-all python-gavodachs
sudo apt-get -f install gavodachs-server
```

With these commands you should end up in the state obtained by a successful package upgrade.

ignoreOn in a rowmaker doesn't seem to work

The most likely reason is that you are testing for the presence of a key that is within the table. This will not work since rowmakers add key->None mapping for all keys missing but metioned in a map (also implicitely via `idmaps`.

If more than one rowmake operate on a source, things get really messy since rowmakers *change* the row dictionaries in place. Maybe this should change at some point, but right now that's the way it is. Thus, you can *never* reliably expect keys used by other tables to be present or absent since you cannot predict the order in which the various table's rowmakers will run.

To fix this, you can check against that key's value being NULL, e.g., like this:

```
<keyIs key="accessURL" value="__NULL__"/>
```

You could also instruct the rowmaker to ignore that key; this would require you to enumerate all rows you want mapped.

Import fails with "Column xy missing" and very few keys

This error is caused by the row validation of the table ingestor – it wants to see values for all table columns, and it's missing one or more. This may be flabbergasting when your grammar yields the fields that are missing here. The reason is simple: You must map them in the rowmaker. If you see this error, you probably wanted to write `idmaps="*"` or something like that in the rowmaker.

Server is Only Visible from the Local Host

When the server is running (`gavo serve start`) and you can access pages from the machine the server runs on just fine, but no other machines can access the server, you run the server with the default web configuration. It tells the server to only bind to the loopback interface (127.0.0.1, a.k.a. localhost).

To fix this, say:

```
[web]
bindAddress:
```

in your /etc/gavo.rc.

Transaction Deadlocking

When gavo imp (or possibly requests to the server) just hangs without consuming CPU but not doing anything useful, it is quite likely that you managed to provoke a deadlock. This happens when you have a database transaction going on a table while trying to access it from the outside.

To give an example:

```
from gavo import base
from gavo import rsc
t = rsc.TableForDef(tableDefForFoo)
q = base.SimpleQuerier().query("select * from foo")
```

This will deadlock if tableDefForFoo actually defines an onDisk table foo. The reason is that instanciating a database table object will create a connection and start a transaction (e.g., to see if the table is actually present on disk).

SimpleQuerier, on the other hand, creates another connection and another transaction. In general, the result of this second transaction will depend on the outcome of the first one. Postgres will notice that and postpone creating the result until the t's transaction if finished. That will never happen with this code.

To diagnose what's happening, it is useful to see the server's idea of what is going on inside itself. The following script (that you might call psdb) will help you:

```
#!/bin/sh
psql gavo << EOF
select procpid, usename, current_query, date_trunc('seconds', query_start::time)
from pg_stat_activity
order by procpid
EOF
```

(this assumes your database is called gavo and you have sufficient rights on that database; it's not hard to figure out the psql command line for other scenarios). This could output something like:

```
 procpid |  usename  |    current_query        | date_trunc
---------+-----------+-------------------------+------------
    9301 | gavoadmin | <IDLE>                  | 16:55:39
    9302 | gavoadmin | <IDLE> in transaction   | 16:55:39
    9303 | gavoadmin | <IDLE> in transaction   | 16:55:39
    9306 | gavoadmin | <IDLE> in transaction   | 16:55:43
    9309 | gavoadmin | SELECT calPar FROM l... | 16:55:43
(5 Zeilen)
```

The procpid is the pid of the process handling the connection. Usually, you will see one running query and possibly quite a few connections that are idle in transaction (which are tables waiting to be fed, etc.).

The query should give you some idea where the deadlock occurs. To escape the deadlock (which, under CPython, will block ^C as well), kill the process trying the query -- this will give you a traceback to the offending instruction. Of course, you will need to become the postgres or root user to do that, so it may be easier to forego the traceback and just kill gavo imp.

To fix such a situation, there are various options. You could commit the table's transaction:

```
from gavo import base
from gavo import rsc
t = rsc.TableForDef(tableDefForFoo)
t.commit()
q = base.SimpleQuerier().query("select * from foo")
```

but that is not usally what you want to do. Much more often, you want to execute the second query in t's transaction. In this case, this could work like this:

```
from gavo import base
from gavo import rsc
t = rsc.TableForDef(tableDefForFoo)
q = base.SimpleQuerier(connection=t.connection).query("select * from foo")
```

Of course, it is not always easy to access the connection object. Note, however, that in most procedure definitions, you have the target data set available as data. If you have that, you can usually obtain the current connection (and thus transaction) via:

```
data.getPrimaryTable().connection
```

-- at least if you designate one of the data's tables as primary through its make elements.

'prodtblAccref' not found in a mapping

You get this error message when you make a table that mixes in //products#table (possibly indirectly, e.g., via SSAP or SIAP mixins) with a grammar that does not use the //products#define row filter.

So, within the grammar, say (at least, see the reference documentation for other parameters for rowgen):

```
<rowfilter procDef="//products#define">
  <bind name="table">"\schema.table"</bind>
</rowfilter>
```

-- substituting dest.table with the actual name of the table fed. The reason why you need to manually give the table is that the grammar doesn't now what table the rows generated end up in. On the other hand, the information needs to be added in the grammar, since it is fed both to your table and the system-wide products table.

I get "Column ssa_dstitle missing" when importing SSA tables

The `//ssap#setMeta` rowmaker application does not directly fill the output rowdict but rather defines new input symbols. This is done to give you a chance to map things set by it, but it means that you must idmap at least all ssa symbols (or map them manually, but that's probably too tedious). So, in the rowmaker definition, you write:

```
<rowmaker idmaps="ssa_*">
```

"unpack requires a string argument of length"

These typically come from a binary grammar parsing from a source with armor=fortran. Then, the input parser delivers data in parcels given by the input file, and the grammar tries to parse it into the fields given in your binaryRecordDef. The error message means that the two don't match.

This can be because the input file is damaged, you forgot to skip some header, but it can also be because you forgot fields or your binaryRecordDef doesn't match the input in some other way.

"resource directory '<whatever>' does not exist"

DaCHS expects each RD to have a "resource directory" that contains input files, auxillary data, etc. Multiple RDs may share a single resource directory.

By default, the resource directory is <inputsDir>/<schema>. If you don't need any auxillary files, the resdir doesn't need to exist. In that case, you'll see the said warning. To suppress it, you could just say:

```
<resource schema="<whatever>" resdir="__system">
```

The `__system` resource directory is used by the built-in RDs and thus should in general exist.

However, the recommended layout is, below `inputsDir`, a subdirectory named like the resource schema, and the RD immediately within that subdirectory. In that case, you don't need a resdir attribute.

Only RDs from below inputsDir may be imported

RDs in DaCHS must reside below `inputsDir` (to figure out what that is on your installation, say `gavo config inputsDir`). The main reason for that restriction is that RDs have identifiers, and these are essentially the inputsDir-relative paths of the file. Out-of-tree RDs just cannot compute this. Therefore, most subcommands that accept file paths just refuse to work when the file in question is not below inputsDir.

Not reloading services RD on server since no admin password available

That's a warning you can get when you run `gavo pub`. The reason for it is that the DaCHS server caches quite a bit of information (e.g., the root page) that may depend on the table of published services (see also Managing Runtime Resources. Therefore, `gavo pub` tries to make the running server discard such caches. To do this, it inspects the `serverURL` config item and tries access a protected resource. Thus, it needs the value of the config setting `adminpasswd` (if set), and that needs to be identical on the machine gavo pub executes on and on whatever serverURL points to.

If anything goes wrong, a warning is emitted. The publication has happened still, but you may need to run `gavo serve reload` on the server to make it visible.

I'm getting "No output columns with these settings." instead of result rows

This is particularly likely to happen with the scs.xml renderer. There, it can happen the the server doesn't even bother to run database queries but instead keeps coming back with an error message `No output columns with these settings.`.

This happens when the "verbosity" (in SCS, this is computed as 10*VERB) of the query is lower than the verbLevel of all the columns. By default, this verbLevel is 20. In order to ensure that a column is returned even with VERB=1, say:

```
<column name=...  verbLevel="1"/>
```

gavo imp dies with Permission denied: u'/home/gavo/logs/dcErrors'

(or something similar). The reason for these typically is that the user that runs `gavo imp` is not in the `gavo` group (actually, whatever [general]gavoGroup says). To fix it, add that user. If that user was, say, fred, you'd say:

```
sudo adduser fred gavo
```

Note that fred will either have to log in and out (or similar) or say `newgrp gavo` after that.

Warnings about popen, md5, etc being deprecated

The python-nevow package that comes with Debian sequeeze is outdated. Install http://docs.g-vo.org/python-nevow_0.11.0-1_all.deb

I'm using reGrammar to parse a file, but no splitting takes place

This mostly happens for input lines like `a|b|c`; the underlying problem is that you're trying to split along regular expression metacharacters. The solution is to escape the the metacharacter. In the example, you wouldn't write:

```
<reGrammar fieldSep="|"> <!-- doesn't work -->
```

but rather:

```
<reGrammar fieldSep="\|"> <!-- doesn't work -->
```

IntegrityError: duplicate key value violates unique constraint "products_pkey"

This happens when you try to import the same "product" twice. There are many possible reasons why this might happen, but the most common (of the non-obvious ones) probably is the use of updating data items with row triggers.

If you say something like:

```
<!-- doesn't work reliably -->
<table id="data" mixin="//products#table"
  ...
<data id="import" updating="True">
  <sources>
    ...
    <ignoreSources fromdb="select accref from my.data"/>
  </sources>
  <fitsProdGrammar...
  <make table="data">
    <rowmaker>
       <ignoreOn name="Skip plates not yet in plate cat">
         <keyMissing key="DATE_OBS"/></ignoreOn>
  ...
```

you're doing it wrong. The reason this yields IntegrityErrors is that if the ignoreOn trigger fires, the row will not be inserted into the table `data`. However, the make feeding the `dc.products` table implicitely inserted by the `//products#table` mixin will not skip an ignored image. So, it will end up in `dc.product`, but on the next import, that source will be tried again – it didn't end up in my.data, which is where ignoreSources takes its file names from –, and boom.

If you feed multiple tables in one data and you need to skip an input row entirely, the only way to do that reliably is to trigger in the grammar, like this:

```
<table id="data" mixin="//products#table"
  ...
<data id="import" updating="True">
  <sources>
    ...
    <ignoreSources fromdb="select accref from my.data"/>
  </sources>
  <fitsProdGrammar...
    <ignoreOn name="Skip plates not yet in plate cat">
       <keyMissing key="DATE_OBS"/></ignoreOn>
  </fitsProdGrammar>
  <make table="data">
  ...
```

gavo init/installation dies with UnicodeDecodeError: 'ascii' codec...

The full signature is something like:

```
File "/usr/lib/python2.7/dist-packages/pyfits/core.py", line 103, in formatwarning
  return unicode(message) + '\n'
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc2 in position 65: ordinal not in range(12
```

This is a bug in pyfits, together with carelessness in passing through error messages on our side. We'll see which side will fix this first; meanwhile, the easy workaround is to set `lc_messages = 'C'` in `postgresql.conf` (e.g., `/etc/postgresql/9.1/main/postgresql.conf` on Debian wheezy). That's probably a good idea anyway since TAP may expose postgres error messages to the user, and these aren't nearly as useful to remote users as to you if they're in your local language.

relation "dc.datalinkjobs" does not exist

This happens when you try to run asynchronous datalink (the dlasync renderer) when you've not created the datalink jobs table. This is not (yet) done automatically on installation since right now we consider async datalink to be a bit of an exotic case. To fix this, run:

```
gavo imp //datalink
```

(some column) may be null but has no explicit null value

These are warnings emitted by the DaCHS' RD parser – since they are warnings, you could ignore them, but you shouldn't.

This is about columns that have no "natural" NULL serialisation in VOTables, mostly integers. Without such a natural NULL, making VOTables out of anything that comes out of these tables can fail under certain circumstances.

There are (at least) two ways to fix this, depending on what's actually going on:

(a) you're sure there are no NULLs in this column. In that case, just add `required="True"`, and the warnings will go away. Note, however, that DaCHS will instruct the database to check that you're not cheating, and an import will fail if your try to put NULLs into such columns.

(b) there are NULLs in this column. In that case, find a value that will work for NULL, i.e., one that is never used as an actual value. "Suspicious" values like 0, -1, -9999 or the like are preferred as this increases the chance that careless programs, formats, or users who ignore a NULL value specification have a chance to catch their error. Then declare that null value like this:

```
<column name="withNull" type="integer"...>
  <values nullLiteral="-9999"/>
</column>
```

Column rave.main.logg_k: Unit dex is not interoperable

The VOUnit standard lets you use essentially arbitrary strings as units – so does DaCHS. VOUnit, however, contains a canon of units VO clients should understand. If DaCHS understands units, you can, for instance, change them on form input and output using the displayUnit displayHint – other programs may allow automatic conversion and similar comforts.

When DaCHS warns that a unit is not interoperable, this means your unit will not be understood in that way. There are cases when that's justified, so it's just a warning, but be sure you understand what you've written and there actually is no interoperable (i.e., using the canonical VOUnits) way to express what you want to say.

Also not that it is an excellent idea to quote free (i.e., non-canonical) units, i.e., write `unit='"Crab"'`. The reason is that in the non-quoted case, VOUnit parsers will try do separate off SI prefixes, such that, e.g., `dex` will be interpreted as dezi-ex, i.e., a tenth of an ex (which happens to actually be a unit, incidentally, although not a canonical VOUnit one).

And yes, dex itself would be a free unit. If you look, quantities given with "dex" as a unit string actually are dimensionless. Our recommendation therefore is to have empty units for them.

Column tab.foo is not a regular ADQL identifier

This is a message you may see when running `gavo val`. It means that the column in question has a name that will get you in trouble as soon as you open the table in question to TAP queries (and trust me, you will sooner or later). Regular ADQL identifiers match the regular expression `[A-Za-z][A-Za-z0-9_]*` with the additional restriction that ADQL reserved words (including terms like distance, size, etc) are not allowed either.

If you see the message, just change the name in question. There's so many nice words that there's really no need to use funny characters in identifiers or hog ADQL reserved words.

If you *must* keep the name anyway, you can prefix it by `quoted/` to make it a delimited identifier. There's madness down that road, though, so don't complain to us if you do that and regret it too late. In particular, you may have a hard time referencing such columns from STC declarations, when creating indices, etc. So: Just don't do it.

Unhandled exception ProgrammingError while importing an obscore table

This typically looks somewhat like this:

```
ProgrammingError: syntax error at or near "/"
LINE 28:       CAST(/RR/V/ AS text) AS pol_states,
                    ^


*** Error: Oops.  Unhandled exception ProgrammingError.

Exception payload: syntax error at or near "/" LINE 28:
CAST(/RR/V/ AS text) AS pol_states,
```

While ProgrammingErrors in general happen whenever an invalid query is sent to the database engine, when they pop up in gavo imp with obscore not far away it almost invariably means that there is a syntax error, most likely forgotten quotes, in the obscore mixin definitions of one of the tables published through obscore. The trick is to figure out which of them causes the trouble.

The most straightforward technique is to take the fragement shown in the error message and look in ivoa._obscoresources like this:

```
$ psql gavo
...
# select tablename
- from ivoa._obscoresources where sqlfragment like '%CAST(/RR/V/%';
     tablename
--------------------
 test.pgs_siaptable
```

You could `gavo purge` the table in question to fix this the raw way, but it's of course much more elegant to just remove the offending piece from _obscoresources:

```
# delete from ivoa._obscoresources where tablename='test.pgs_siaptable';
```

Then fix the underlying problem – in this case that was replacing:

```
<mixin
polStates="/RR/V/"
...
```

with:

```
<mixin
polStates="'/RR/V/'"
...
```

– and re-import the obscore meta; you'll usually use `gavo imp -m && gavo imp //obscore` for that (see also updating obscore)

cert already in hash table

Under circumstance we've not quite understood yet either, in Debian stretch DaCHS may dump a long traceback on an error like this:

```
cryptography.exceptions.InternalError: Unknown OpenSSL error. This error is commonly encountered
```

This is a race condition deep within python-cryptography, which is in turn is several levels below nevow; what's worse, this happens during import, so even monkeypatching is at least very difficult.

We currently try to work around it by importing the racing component before any threads can occur. The hack will be in DaCHS 1.0 and the 1.0.2 beta. It's a hack, though, and it's possible it won't work for you. Let us know if you hit this.

dachs init fails with "type spoint does not exist"

This always means that the pgsphere extension could not be loaded, and DaCHS can no longer do without it. Actually, we could try to make it, but you really need pg_sphere in almost all installations, so it's better to fix this than to work around it.

Unfortunately, there is any number of reasons for a missing pgsphere.

If, for instance, you see this error message and have installed DaCHS from tarball or svn, with manual dependency management, just install the pgsphere postgres extension (and, while you're at it, get q3c, too).

If this happens while installing the Debian package, in all likelihood DaCHS is not talking to the postgres version in thinks it is. This very typically happens if you already have an older postgres version on the box. Unless you're sure you know what you're doing, just perform an upgrade to the version DaCHS wants – see howDoI.html#upgrade-the-database-engine. If you'd need to downgrade, that's trouble. Complain to the dachs-support mailing list – essentially, someone will have to build a pgsphere package for your postgres version.

relation "ivoa._obscoresources" does not exist

This happens when you try to import an obscore-published table (these mix in something like `//obscore#publish-whatever`) without having created the obscore table itself. The fix is easy: Either remove the mixin if you don't want the obscore publication (which would be odd for production data) or, more typically, create the obscore table:

```
dachs imp //obscore
```

duplicate key value violates unique constraint "tables_pkey"

This typically happens on `dachs imp`. The immediate reason is that `dachs imp` tries to insert a metadata row for one of the tables it just created into the `dc.tablemeta` system table, but a row for that name is already present. For instance, if you're importing into `arihip.main`, DaCHS would like to note that the new table's definition can be found at `arihip/q#main`. Now, if `dc.tablemeta` already says `arihip.main` was defined in `quicktest/q#main`, there's a problem that DaCHS can't resolve by itself.

90% of the time, the underlying reason is that you renamed an RD (or a resource directory). Since the identifier of an RD (the RD id) is just is relative path of the RD to the inputs directory (minus the `.rd`), and the RD id is used in many places in DaCHS, you have to be careful when you do that (essentially: `dachs drop --all old/rd; mv old new; dachs imp new/rd`).

If you're seeing the above message, it's already too late for that careful way. The simple way to repair things nevertheless is to look for the table name (it should be given in the DETAILS of the error message) and simply tell DaCHS to forget all about that table:

```
dachs purge arigfh.main
```

This might leave other traces of the renamed RD in the system, which might lead to trouble later. If you want to be a bit more throrough, figure out the RD id of the vanished RD by running `psql gavo` and asking something like:

```
select sourcerd from dc.tablemeta where tablename='arihip.main'
```

This will give you the RD id of the RD that magically vanished, and you can then say:

```
dachs drop -f old/rdid
```

DaCHS will then hunt down all traces of the old RD and delete them.

Don't do this without an acute need; such radical measures will clean up DaCHS' mind, but in a connected society, amnesia can be a strain on the rest of the society. In the VO case, `dachs drop -f` might, for instance, cause stale registry records if you had registered services inside of the RD you force-drop.