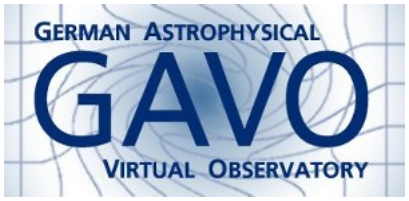

Fig. 1

Fig. 2

1. Anatomy of an RD

(cf. Fig. 1)

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

(cf. Fig. 2)

RDs are DaCHS' way of describing

- General metadata (Registry!)
- Table structure
- Ingestions rules
- Services
- Regression tests

Note: RDs are *not* a VO standard, so other tools use other means of describing their resources.

2. Resource Descriptors

A single DaCHS server usually publishes multiple "resources".

Each of these has a resource directory and a descriptor (RD) in it. "Publishing" largely means: Write an RD.

Examples:

- Most of the ones at the GAVO Data Center[1] are public: http://svn.ari.uni-heidelberg.de/svn/gavo/hdinputs
- For EPN: http://svn.ari.uni-heidelberg.de/svn/gavo/hdinputs/titan/ (when data products are served by third parties)
- and http://docs.g-vo.org/epntap-example.tar.gz (when data products are to be served by DaCHS itself)updates

[1] `http://dc.g-vo.org`

3. Global metadata

This lets DaCHS build information pages and Registry records

```
<resource schema="emi">
  <meta name="title">VLBI images of Lockman Hole radio sources</meta>
  <meta name="description" format="plain">
    These are 1.4GHz Very Long Baseline Interferometry
    images of 532 radio...
  <meta name="creator">
    <meta name="name">Middelberg, E.</meta>...
  <meta name="source">2013A&amp;amp;A...551A..97M</meta>
  <meta name="coverage">
    <meta name="profile">Circle ICRS 163 57.5 1
      SpectralInterval TOPOCENTER 1.318 1.446...
    <meta name="waveband">Radio</meta>
  </meta>
  <meta name="creationDate">2013-04-09T12:57:26Z</meta>
  <meta name="subject">Techniques: interferometric</meta>
  <meta name="facility"> Very Long Baseline Array, operated
    by The National Radio Astronomy Observatory</meta>
```

RDs are written in XML and thus have a root element, the opening tag of which is shown here, too. It has a mandatory attribute called `schema`, which gives the DB schema and (here also) the name of the directory below DaCHS' inputs where it will look for data and other files (the "resource directory").

All this is taken from DaCHS' tutorial; to get the full RD and data to play with, refer to

http://docs.g-vo.org/DaCHS/tutorial.html#id3

4. Table Structure

Most DaCHS services work on top of a DB table, in this case containing the image metadata. Its definition looks like this:

```
  <table id="main" onDisk="True">
    <mixin>//siap#pgs</mixin>
    <column name="object" type="text"
      ucd="meta.id"
      tablehead="Object"
      description="Source name as in
        2009MNRAS.397..281I (VizieR J/MNRAS/397/281)"
      verbLevel="1"/>
    ....
```

You define columns with rich metadata. most VO protocols are accompanied by a mixin, sets of columns, indices, and other table features required to implement the protocol.

Here, it's SIAP (image access) based on pgSphere.

You can, however, just enumerate your columns manually, too.

If you look at the sample RD, you'll notice a second mixin that exports the data to another table, obscore, that lets users query the data holdings via a standard table structure and TAP. That shows two things: a table can have multiple mixins, and mixins can have parameters.

The reference documentation shows what mixins are there and what their parameters are (of course, you can define mixins of your own; the element content of mixin is just a DaCHS-internal reference to an RD.

5. Ingestion Rules

These consist of data search paths, a "grammar", and mapping rules:

```
<data id="import_main">
  <sources recurse="True">
    <pattern>data/*.fits</pattern>
  <fitsProdGrammar qnd="True">
    <rowfilter procDef="__system__/products#define">
      <bind key="table">"emi.main"</bind>
  <make table="main" >
    <rowmaker id="gen_rmk" idmaps="object, obsra, obsdec">
      <apply procDef="//siap#computePGS"/>
      <apply procDef="//siap#setMeta">
        <bind name="bandpassLo">0.207</bind>
        ...
      <apply name="fixObjectName"> ...
            import csv
            with open(rd.getAbsPath("res/namemap.csv")) as f:
              nameMap = dict(csv.reader(f))...
      <map key="weighting">\inputRelativePath.split("_")[-1][:-5]</map>
```

Mapping data is a complex business that frequently needs procedural instructions. For maintainability, being as prodecural as possible is highly desirable, though. DaCHS tries to satisfy both, letting you define your own "applys" (which can contain python code) and use python expressions in mappings, but giving you sensible defaults through features like `idmaps` and predefined applys; here, it's computing coverage and such from WCS (`//siap#computePGS`) and setting all kinds of non-WCS metadata (`//siap#setMeta`).

6. Grammars

Grammars turn external data into a sequence for dictionaries. DaCHS comes with grammars for

- FITS headers and data
- PDS headers
- All kinds of text formats
- ... and lots more

When nothing fits: Provide your own python code in customGrammar (extra file) or embeddedGrammar (in the RD).

When you have millions of rows: use "boosters" (directGrammars)

7. Services

Services have a core describing the interaction with the data, and service elements giving the interaction with the user:

```
  <dbCore id="imcore" queriedTable="main">
    <condDesc original="//siap#protoInput"/>
    <condDesc original="//siap#humanInput"/>
    <condDesc buildFrom="dateObs"/>

  <service id="browse" allowed="form" core="imcore">
    <outputTable autoCols="accref, accsize, object, obsra, obsdec, dateObs,
      imageTitle"/>

  <service id="s" allowed="form,siap.xml" core="imcore">
    <meta name="sia.type">Pointed</meta>
    <meta name="testQuery.pos.ra">163.3</meta>
    <publish render="siap.xml" sets="ivo_managed"/>
    <publish render="form" sets="ivo_managed,local" service="browse"/>
```

This is a dbCore, which is fairly typical: It's just a query against `queriedTable`. Here, it is used by two services, a web-based one for interaction with web browsers, and as standard SIAP one with additional metadata.

8. Regression Tests

Things break. It's a good idea to automate diagnosis. In DaCHS:

```
<regTest title="Gaia DR1 has expected data">
  <url parSet="TAP"
    QUERY="select * from gaia.dr1 where source_id=1978292722869983488"
      >/tap/sync</url>
  <code>
    row = self.getFirstVOTableRow()
    self.assertAlmostEqual(row["pmra_pmdec_corr"], 0.9385120272636414)
    self.assertAlmostEqual(row["pmdec_error"], 1.12316)
    self.assertEqual(row["astrometric_delta_q"], None)
  </code>
</regTest>
```

Run tests from one RD `dachs test q`, for all in your data center with `dachs test ALL`.

9. Documentation

As usual, too much and too little.

- Base URL: http://docs.g-vo.org/DaCHS/ (including reference manual)
- Mirror: http://dachs-doc.readthedocs.io/index.html (no ref manual so far, but interface to github issues)
- Reference for RD elements as they're used in our DC: http://docs.g-vo.org/DaCHS/elemref.html
- Support mailing list: http://lists.g-vo.org/cgi-bin/mailman/listinfo/dachs-support (needs more traffic!)

If you run DaCHS please subscribe to
http://lists.g-vo.org/cgi-bin/mailman/listinfo/dachs-users