

A new VOTable library in Rust (and more!)

F.-X. Pineau¹, T. Dumortier¹

¹Centre de Données astronomiques de Strasbourg

11th May 2023



□ VOTable

- VOTable: the **first IVOA standard (2003)**
 - **XML hype** at the time
 - Precursor: Astrores
 - XML for metadata
 - ASCII/CSV for table rows
- [IVOA VOTableSoftware wiki](#) page (up-to-date ?)
- Existing libraries (I am aware of, very limited!)
 - STIL (TOCPAT/STILTS): **Java**
 - Astropy: **Python**
 - Savot: Java, maintained ?
 - votable.js: JavaScript, maintained ?
 - votpp: C++, updated in 2014 (maintained?)
 - ...

□ A new library?!


- **Motivations** for a new library in Rust
 - **Needed for other CDS Rust tools**
 - **Aladin Lite v3** core is in Rust
 - ExXmatch is in Rust
 - **VizieR large catalogues** query code (switching to Rust)
 - ...
 - Usual argument: **write once, re-use in various places**
 - natively in Rust projects
 - Python wrapper
 - JavaScript/WASM wrapper
 - ...
 - **VOTable as a “model”** (de)serializable in **various formats?**
 - XML: current VOTable format
 - JSON: for web browsers
 - TOML: to update metadata manually
 - YAML: why not?
 - **preservation** in round-trip transformations possible?

□ Build functionality

- Build VOTable programmatically
 - **in memory objects** for each tag
 - but data can be in memory **or** generated on-the-fly
 - **coherent** with the schema **by construction**
 - but (e.g.) fields vs actual data not checked (yet?)

Struct votable::field::Field 

```
pub struct Field {  
    [-]  
    pub id: Option<String>,  
    pub name: String,  
    pub datatype: Datatype,  
    pub unit: Option<String>,  
    pub precision: Option<Precision>,  
    pub width: Option<u16>,  
    pub xtype: Option<String>,  
    pub ref_: Option<String>,  
    pub ucd: Option<String>,  
    pub utype: Option<String>,  
    pub arraysize: Option<String>,  
    pub extra: HashMap<String, Value>,  
    pub description: Option<Description>,  
    pub values: Option<Values>,  
    pub links: Vec<Link>,  
}
```

Enum votable::datatype::Datatype 

```
pub enum Datatype {  
    Logical,  
    Bit,  
    Byte,  
    ShortInt,  
    Int,  
    LongInt,  
    CharASCII,  
    CharUnicode,  
    Float,  
    Double,  
    ComplexFloat,  
    ComplexDouble,  
}
```

□ Programmatic build 1/3

```
let table = Table::new()
    .set_name("V/147/sdss12")
    .set_description("SDSS photometric catalog".into())
    .push_field(
        Field::new("RA_ICRS", Datatype::Double)
            .set_unit("deg")
            .set_ucd("pos.eq.ra;meta.main")
            .set_width(10)
            .set_precision(Precision::new_dec(6))
            .set_description("Right Ascension (ICRS)".into())
    ).push_field(
        ...
    ).set_data(Data::new_empty().set_tabledata(...));
```

□ Programmatic build 2/3

```
let resource = Resource::default()
    .set_id("yCat_17011219")
    .set_name("J/ApJ/701/1219")
    .set_description(r#"Photometric catalog ..."#.into())
    .push_coosys(
        CooSys::new("J2000", System::new_default_eq_fk5())
    ).push_coosys(
        CooSys::new(
            "J2015.5",
            System::new_icrs().set_epoch(2015.5))
    ).push_table(table)
    .push_post_info(
        Info::new("QUERY_STATUS", "OVERFLOW")
            .set_content("Truncated result")
    );
```

□ Programmatic build 3/3

```
let votable = VOTable::new(resource)
    .set_id("my_votable")
    .set_version(Version::V1_4)
    .set_description(r#"VizieR Astronomical Server"#.into())
    .push_info(
        Info::new("votable-version", "1.99+ (14-Oct-2013)")
            .set_id("VERSION")
    );
```

□ Parsing a VOTable

Build from existing VOTable, multiple ways to parse a VOTable

- one hybrid DOM / SAX(push) / StAX(pull) like mode
 - parse everything except table data
 - (similar to the M. Taylor choice in TOPAT/STILT ?)
 - **generic class to delegate table data parsing**
 - either **built-in: load everything in memory**
 - **either each row is an array of String** (not for base64)
 - **or each row is an array of objects** (f64, int, ...)
 - or implement your own table data parser
 - SAX call to table data parser
 - table data parser rely on StAX parsing ([quick_xml](#) reader)
 - allows for **streaming**
- one hybrid DOM/StAX like parser
 - **iterator on table objects** providing:
 - access to the **table metadata** objects
 - + **iterator on table rows**
 - also allows for **streaming**
 - (current limitation: no access to RESOURCE metadata)

□ (De)serialization

- (De)serialize VOTable/JSON/TOML/YAML to/from memory objects (built-in)

```
fn from_ivoa_xml_reader>(…) -> …
```

```
fn from_json_reader(…) -> …
```

```
fn from_toml_reader(…) -> …
```

```
fn from_yaml_reader(…) -> …
```

```
fn to_ivoa_xml_writer>(…) -> …
```

```
fn to_json_writer(…) -> …
```

```
fn to_toml_writer(…) -> …
```

```
fn to_yaml_writer(…) -> …
```

```
…
```

□ VOTCli and MOCWasm

- Tools to convert (memory manageable) VOTables in XML/JSON/TOML/YAML
 - from the command line: **VOTCli**
 - pre-compiled for various architecture
 - .deb packages
 - pypi: <https://pypi.org/project/votable-cli/>
 - in a Web Browser: **VOTWasm**
 - JS/WebAssembly library, 100% Rust (wasm-bindgen)
 - rely on intermediary JsObject
 - available in [github release](#)



VOTcli

```
fxpineau@cds-dev-fxp-7300:~$ vot --help
Command-line to convert IVOA VOTables in XML, JSON, YAML and TOML

Usage: vot [OPTIONS] <INPUT_FMT> <OUTPUT_FMT>

Arguments:
  <INPUT_FMT>  Format of the input document ('xml', 'json', 'yaml' or 'toml')
  <OUTPUT_FMT> Format of the output document ('xml', 'json', 'yaml' or 'toml')

Options:
  -i, --input <FILE>  Input file (else read from stdin)
  -o, --output <FILE> Output file (else write to stdout)
  -p, --pretty          Pretty print (for JSON and TOML)
  -h, --help           Print help
  -V, --version        Print version
```

```
> vot --input gaia_dr3.b64.vot xml yaml
> vot -i vot.xml -o vot.toml xml toml --pretty
> cat b64.vot | \
>   vot xml json --pretty 2> /dev/null | more
```

Missing: options to switch TABLEDATA/BINARY/BINARY2

□ VOTCli Example

- Convert a Vizier output VOTable into TOML
 - in: 2.5 MB XML (TABLEDATA) file
 - out: 2.0 MB TOML file
 - conversion time: 70ms

```
fxpineau@cds-dev-fxp-7300:~$ time vot -i 2p5mega.vot -o vot.toml xml toml --pretty 2> /dev/null
real    0m0,067s
user    0m0,067s
sys     0m0,000s
fxpineau@cds-dev-fxp-7300:~$ head -10 vot.toml
[votable]
version = '1.4'
"xmlns:xsi" = 'http://www.w3.org/2001/XMLSchema-instance'
xmlns = 'http://www.ivoa.net/xml/VOTable/v1.3'
"xsi:schemaLocation" = 'http://www.ivoa.net/xml/VOTable/v1.3 http://www.ivoa.net/xml/VOTable/v1.3'
description = ''

Vizier Astronomical Server vizier.u-strasbg.fr
Date: 2020-11-17T10:05:52 [v1.99+ (14-Oct-2013)]
Explanations and Statistics of UCDS: See LINK below
```

□ VOTWasm

- Download JS/Wasm files from [github releases](#)
- Available functions

```
vot.fromXML(string) -> JsObject  
vot.toXML(JsObject) -> String  
vot.fromJSON(string) -> JsObject  
vot.toJSON(JsObject) -> String  
vot.fromTOML(string) -> JsObject  
vot.toTOML(JsObject) -> String  
vot.fromYAML(string) -> JsObject  
vot.toYAML(JsObject) -> String
```

- No API to build a conform JsObject
 - error in case of not -conform JsObject

□ Rust Serde!

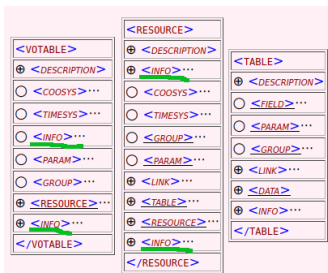
- Internally the magic happens thanks to `serde`

```
#[derive(Clone, Debug, serde::Serialize, serde::Deserialize)]
pub struct Group {
    // attributes
    #[serde(rename = "ID", skip_serializing_if = "Option::is_none")]
    id: Option<String>,
    #[serde(skip_serializing_if = "Option::is_none")]
    name: Option<String>,
    #[serde(rename = "ref", skip_serializing_if = "Option::is_none")]
    ref_: Option<String>,
    #[serde(skip_serializing_if = "Option::is_none")]
    ucd: Option<String>,
    #[serde(skip_serializing_if = "Option::is_none")]
    utype: Option<String>,
    // Sub-elems
    #[serde(skip_serializing_if = "Option::is_none")]
    description: Option<Description>,
    #[serde(default, skip_serializing_if = "Vec::is_empty")]
    elems: Vec<GroupElem>,
}
```

Figure 1: Serde annotations on GROUP

□ Design choices 1

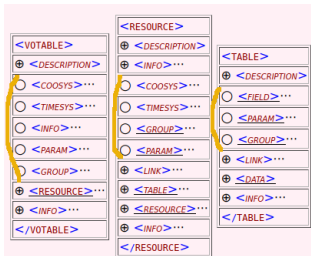
- The INFO problem?
 - “The INFO element may occur before the closing tags /TABLE and /RESOURCE and /VOTABLE (enables post-operational diagnostics)”
 - one solution: make a difference between info and post-info



```
pub struct Resource<C: TableDataContent> {  
    pub id: Option<String>,  
    pub name: Option<String>,  
    pub type_: Option<String>,  
    pub utype: Option<String>,  
    pub extra: HashMap<String, Value>,  
    pub description: Option<Description>,  
    pub infos: Vec<Info>,  
    pub elems: Vec<ResourceElem>,  
    pub links: Vec<Link>,  
    pub tables: Vec<Table<C>>,  
    pub resources: Vec<Resource<C>>,  
    pub post_infos: Vec<Info>,  
}
```

□ Design choices 2

- Multiple possible sub-elements at a given place
 - one solution: group them in a same enum



```
pub struct Resource<C: TableDataContent> {  
    pub id: Option<String>,  
    pub name: Option<String>,  
    pub type_: Option<String>,  
    pub utype: Option<String>,  
    pub extra: HashMap<String, Value>,  
    pub description: Option<Description>,  
    pub infos: Vec<Info>,  
    pub elems: Vec<ResourceElem>,  
    pub links: Vec<Link>,  
    pub tables: Vec<Table<C>>,  
    pub resources: Vec<Resource<C>>,  
    pub post_infos: Vec<Info>,  
}
```

```
pub enum ResourceElem {  
    CooSys(CooSys),  
    TimeSys(TimeSys),  
    Group(Group),  
    Param(Param),  
}
```


□ Design choices 3

(I am actually not sure any more about this one)

- GROUP:
 - cannot contain FIELDRef in VOTABLE and RESOURCE
 - may contain FIELDRef in TABLE
- one solution: 2 different structures/classes (internally)
 - **Group**
 - **TableGroup**

```
pub enum GroupElem {  
    ParamRef(ParamRef),  
    Param(Param),  
    Group(Group),  
}
```

```
pub enum TableGroupElem {  
    FieldRef(FieldRef),  
    ParamRef(ParamRef),  
    Param(Param),  
    TableGroup(TableGroup),  
}
```

Other debatable choices 1/3

- XML **TAGs** in **lower case**
- **'s'** after a TAG name
 - if multiple occurrence allowed (JSON array)
- **'elems'** + **'elem_type'** + camel case class/struct name
 - when multiple TAGs are possible

```
"votable": {
  "version": "1.4",
  ...
  "description": "VizieR Astronomical Server ...",
  "elems": [
    {
      "elem_type": "Info",
      "ID": "VERSION",
      "name": "votable-version",
      "value": "1.99+ (14-Oct-2013)"
    }, ...
  ],
  "resources": [
    {
      "ID": "yCat_74692163",
      "name": "J/MNRAS/469/2163",
      "description": "JCMT Plane Survey. first complete data",
      "elems": [
        {
          "elem_type": "CooSys",
          "ID": "G",
          "system": "galactic"
        }, ...
      ],
      "tables": [
        {
          "id": "J_MNRAS_469_2163_table2",
          "name": "J/MNRAS/469/2163/table2",
          "description": "JPSR1 compact source catalogue",
          "elems": [
            {
              "elem_type": "Field",
              "name": "RAJ2000",
              "datatype": "double",
            }
          ]
        }
      ]
    }
  ]
}
```

Other debatable choices 2/3

- **'data_type'** to differentiate
 - TableData
 - Binary
 - Binary2
 - Fits
- **'rows'** for the TableData content

```
],  
  "data": {  
    "data_type": "TableData",  
    "rows": [  
      [  
        270.46713,  
        -23.32183,  
        1,  
        "JPSG006.687-00.294",  
        "JCMTLSPJ180152.1-231918",  
        6.687,  
        -0.294,  
        6.688,  
        -0.292,  
        16,  
        5,  
        120,  
        17,  
        0.484,  
        0.024,  
        1.166,  
        0.124,  
        17.04,  
        270.4671,  
        -23.3218  
      ],  
      [  
        270.54551,  
        -23.29028,  
        1,  
        "JPSG006.687-00.294",  
        "JCMTLSPJ180152.1-231918",  
        6.687,  
        -0.294,  
        6.688,  
        -0.292,  
        16,  
        5,  
        120,  
        17,  
        0.484,  
        0.024,  
        1.166,  
        0.124,  
        17.04,  
        270.5455,  
        -23.29028  
      ]  
    ]  
  }  
}
```

Other debatable choices 3/3

- **'infos'** != **'post_infos'**
- **'content'** to store an XML TAG content having possible attributes

```
    "post_infos": [  
      {  
        "name": "matches",  
        "value": "7813",  
        "content": "matching records"  
      },  
      {  
        "name": "Warning",  
        "value": "No center provided++++"  
      },  
      {  
        "name": "Warning",  
        "value": "skip recno in all column"  
      },  
      {  
        "name": "Warning",  
        "value": "increase the precision f"  
      }  
    ]  
  }  
}
```

□ Other alternatives for JSON

- Python package `xml2json`?
- Python `lxml` + `xml2dic` + `json` tested by L. Michel, see [votable-rust README file](#)
 - pros:
 - few lines of Python
 - generic (for all XML files)
 - cons:
 - order of sub-elements is lost (e.g. no INFO/POST-INFO; order important in DALI)
 - no in-memory structure
 - no JON to XML VOTable conversion
- ... ?
- In any case: we have to agree on choices to be inter-operable (do we want to)!!

Remark on Rust

- Writing each tag code is TEDIOUS!!
 - lot of duplicated/almost duplicated code
 - setters, getters, push element, test if null, ...
- But Rust has declaratives macros : objects boilerplate code
- Serde's procedural macros: serialization boilerplate code!

```
// attributes
impl_builder_opt_string_attr!(id);
impl_builder_opt_string_attr!(unit);
impl_builder_opt_attr!(precision, Precision);
impl_builder_opt_attr!(width, u16);
impl_builder_opt_string_attr!(xtype);
impl_builder_opt_string_attr!(ref_, ref);
impl_builder_opt_string_attr!(ucd);
impl_builder_opt_string_attr!(utype);
impl_builder_opt_string_attr!(arraysize);
// extra attributes
impl_builder_insert_extra!();
// sub-elements
impl_builder_opt_attr!(description, Description);
impl_builder_opt_attr!(values, Values);
impl_builder_push!(Link);
```

```
pub fn set_id<I: Into<String>>(mut self, id: I) -> Self {
    self.id = Some(id.into());
    self
}
pub fn set_unit<I: Into<String>>(mut self, unit: I) -> Self {
    self.unit = Some(unit.into());
    self
}
pub fn set_precision(mut self, precision: Precision) -> Self {
    self.precision = Some(precision);
    self
}
pub fn set_width(mut self, width: u16) -> Self {
    self.width = Some(width);
    self
}
pub fn set_xtype<I: Into<String>>(mut self, xtype: I) -> Self {
    self.xtype = Some(xtype.into());
    self
}
pub fn set_ref<I: Into<String>>(mut self, ref_: I) -> Self {
    self.ref_ = Some(ref_.into());
    self
}
pub fn set_ucd<I: Into<String>>(mut self, ucd: I) -> Self {
    self.ucd = Some(ucd.into());
    self
}
pub fn set_utype<I: Into<String>>(mut self, utype: I) -> Self {
```

□ Final word

- VOTable Lib Rust:
 - repo <https://github.com/cds-astro/cds-votable-rust>
 - dual MIT/Apache 2.0 licensing
 - crates.io: <https://crates.io/crates/votable>
 - doc to be improved!!
 - **young** but **in production** in **Aladin Lite v3!**
 - bugs will be fixed quickly (send us VOTables!)
 - **but** parts are **still exploratory**: several way to read/write a VOTable (even for streaming)
 - async version?
- What about the **VOTable as model**?
 - who already did that?
 - different choices: **share experience!**
 - **should we agree** on VOTable JSON/TOML/YAML formats?
 - binary format (with FITS random access property) desirable?