MOC lib Rust and its derivatives: MOCPy, MOCWasm, MOCCli and MOCSet

F.-X. Pineau[1], M. Baumann[1], T. Boch[1], P. Fernique[1]

[1] Centre de Données astronomiques de Strasbourg

27th April 2022

□ Introduction

MOC Lib Rust: a Rust implementation of the IVOA MOC 2.0 standard.

- Idea: one core library for multiple targets
  - Python users (MOCPy)
  - Standalone command line tool (MOCCli)
  - Web Browsers applications (MOCWasm)
  - PostgreSQL functions?
  - ...
- Mainly 3 possibles languages: C, C++, **Rust**
  - compiled, no runtime, no GC
  - existing wrappers to interact with various other languages

History 1/2

- MOC Java (2010-2022, P. Fernique)
- MOCPy early dev in pure Python (2015-2018, T. Boch)
  - See 2015 InterOp presentation by T. Boch
  - Target the python users community
- MOCPy heavy dev (2018-2020, M. Baumann)
  - See May 2019 InterOp presentation by M. Baumann
  - See October 2019 InterOp presentation by M. Baumann
  - Introduction of Rust with PyO3 to improve perfs
  - Uses cdshealpix-rust and its python wrapper
  - CI based on AppVeyor and Travis
  - Astropy affiliated package
  - Implementation of MOC pre 2.0 (T-MOC and ST-MOC)

□ History 2/2

- MOCPy third development phase (2021-2022, F.X. Pineau)
  - New CI based on github Actions
  - Add genericity to be MOC 2.0 compatible
  - Add functionalities on Rust side
    - serialization/deserialization
    - direct calls to cds-healpix-rust
    - . . .
  - Improve some perfs (and, or, extend, contract, contains, uniq2range, . . .)
  - Add new functionalities (lazy operations, . . .)
  - Add Gravitational Waves community requirements (from G. Greco)
- Standalone Rust library (MOC lib Rust)
  - still a thin Rust wrapper in MOCPy
  - MOCWasm, MOCCli, . . .

□ MOC lib Rust functionalities

- serialize/deserialize S/T/ST-MOCs
  - ASCII, JSON, (gzipped) FITS
- build S-MOCs from:
  - cone, elliptical cone, box, zone, polygon, ring
  - list of positions, list of large/small cones
  - multi-order map, skymap
- multi logical or to build a S-MOC from multiple S-MOCs
- logical operations (both standard and lazy):
  - not, and, or, xor, minus
  - lazy = operations on iterators => streaming mode
  - generic: same code for Space u32 or Time u64
- other operations
  - degrade, extend, contract, external/internal border
  - split into disjoint MOCs
- build T-MOCs, ST-MOCs
- missing: MOC from images, MOC visualization

MOCPy

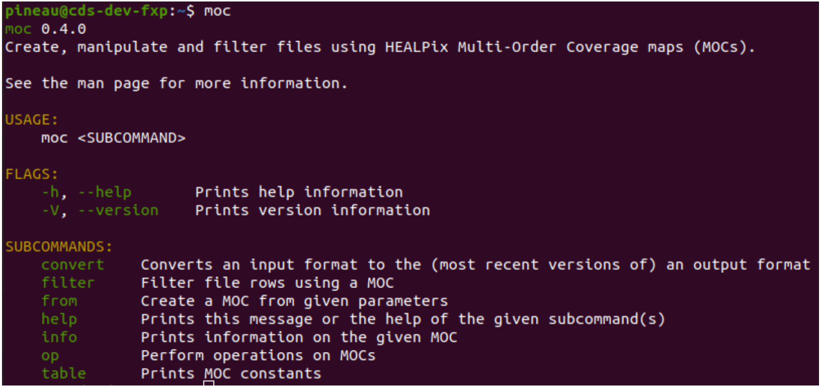
MOCPy is a Python library to load, parse and manipulate MOCs

- Previous IVOA presentations
  - See 2015 InterOp presentation by T. Boch
  - See May 2019 InterOp presentation by M. Baumann
  - See Oct. 2019 InterOp presentation by M. Baumann
- Astropy affiliated package
- Available in both pypi and conda
  - `pip install --upgrade mocpy`
  - `conda install mocpy`
- On GitHub: `https://github.com/cds-astro/mocpy`
  - BSD 3 license
  - documentation: `https://cds-astro.github.io/mocpy/`
- Rust/Python binding based on PyO3

MOCCli

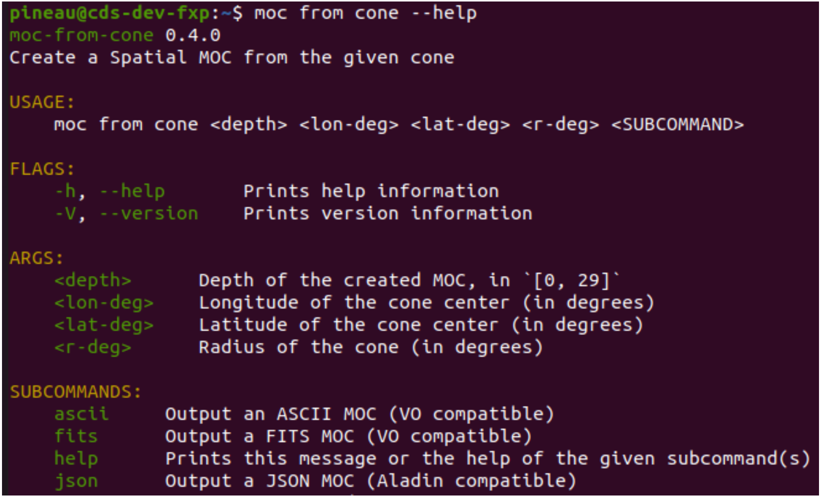
- Manipulate MOCs without installing Python
- Document the usage of MOC Lib Rust capabilities
- Open source: on GitHub, Apache 2.0 + MIT dual licensing
- See `https://github.com/cds-astro/cds-moc-rust/releases`
  - pre-compiled binaries for Linux, MacOS and Windows (14 MB)
  - both 32 and 64 bits .deb packages
- Based on the structopt crate to
  - parse from the command line
  - generate the usage messages

MOCCli usage screenshot



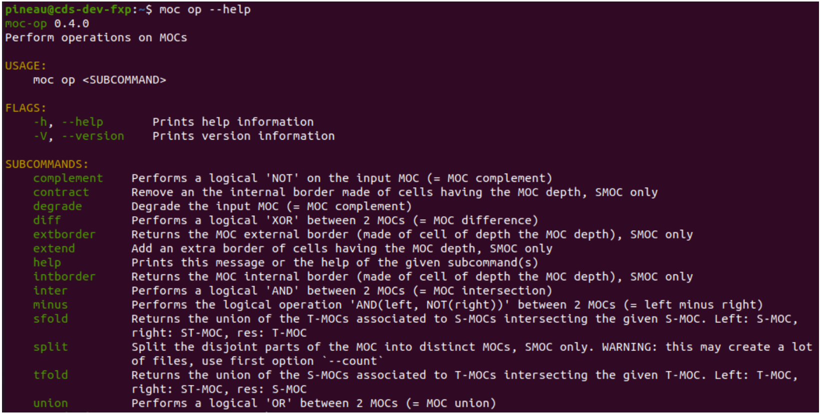
```
pineau@cds-dev-fxp:~$ moc
moc 0.4.0
Create, manipulate and filter files using HEALPix Multi-Order Coverage maps (MOCs).

See the man page for more information.

USAGE:
    moc <SUBCOMMAND>

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

SUBCOMMANDS:
    convert    Converts an input format to the (most recent versions of) an output format
    filter     Filter file rows using a MOC
    from       Create a MOC from given parameters
    help       Prints this message or the help of the given subcommand(s)
    info       Prints information on the given MOC
    op         Perform operations on MOCs
    table      Prints MOC constants
```

MOCCli usage screenshot



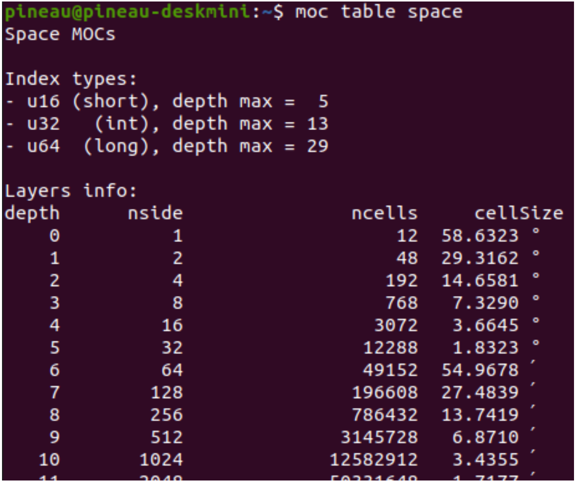
```
pineau@cds-dev-fxp:~$ moc from cone --help
moc-from-cone 0.4.0
Create a Spatial MOC from the given cone

USAGE:
    moc from cone <depth> <lon-deg> <lat-deg> <r-deg> <SUBCOMMAND>

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

ARGS:
    <depth>      Depth of the created MOC, in `[0, 29]`
    <lon-deg>    Longitude of the cone center (in degrees)
    <lat-deg>    Latitude of the cone center (in degrees)
    <r-deg>      Radius of the cone (in degrees)

SUBCOMMANDS:
    ascii     Output an ASCII MOC (VO compatible)
    fits      Output a FITS MOC (VO compatible)
    help      Prints this message or the help of the given subcommand(s)
    json      Output a JSON MOC (Aladin compatible)
```

MOCCli usage screenshot



```
pineau@cds-dev-fxp:~$ moc op --help
moc-op 0.4.0
Perform operations on MOCs

USAGE:
    moc op <SUBCOMMAND>

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

SUBCOMMANDS:
    complement    Performs a logical 'NOT' on the input MOC (= MOC complement)
    contract      Remove an the internal border made of cells having the MOC depth, SMOC only
    degrade       Degrade the input MOC (= MOC complement)
    diff          Performs a logical 'XOR' between 2 MOCs (= MOC difference)
    extborder     Returns the MOC external border (made of cell of depth the MOC depth), SMOC only
    extend        Add an extra border of cells having the MOC depth, SMOC only
    help          Prints this message or the help of the given subcommand(s)
    intborder     Returns the MOC internal border (made of cell of depth the MOC depth), SMOC only
    inter         Performs a logical 'AND' between 2 MOCs (= MOC intersection)
    minus         Performs the logical operation 'AND(left, NOT(right))' between 2 MOCs (= left minus right)
    sfold         Returns the union of the T-MOCs associated to S-MOCs intersecting the given S-MOC. Left: S-MOC,
                  right: ST-MOC, res: T-MOC
    split         Split the disjoint parts of the MOC into distinct MOCs, SMOC only. WARNING: this may create a lot
                  of files, use first option `--count`
    tfold         Returns the union of the S-MOCs associated to T-MOCs intersecting the given T-MOC. Left: T-MOC,
                  right: ST-MOC, res: S-MOC
    union         Performs a logical 'OR' between 2 MOCs (= MOC union)
```

MOCCli examples

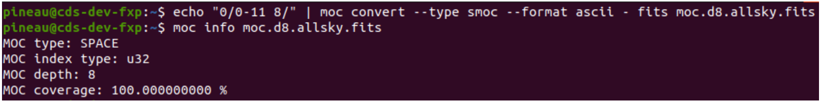


```
pineau@pineau-deskmini:~$ moc table space
Space MOCs

Index types:
- u16 (short), depth max =  5
- u32   (int), depth max = 13
- u64  (long), depth max = 29

Layers info:
depth      nside                   ncells     cellSize
    0          1                       12   58.6323 °
    1          2                       48   29.3162 °
    2          4                      192   14.6581 °
    3          8                      768    7.3290 °
    4         16                     3072    3.6645 °
    5         32                    12288    1.8323 °
    6         64                    49152   54.9678 ′
    7        128                   196608   27.4839 ′
    8        256                   786432   13.7419 ′
    9        512                  3145728    6.8710 ′
   10       1024                 12582912    3.4355 ′
   11       2048                 50331648    1.7177 ′
```

Figure 1: HEALPix reminder

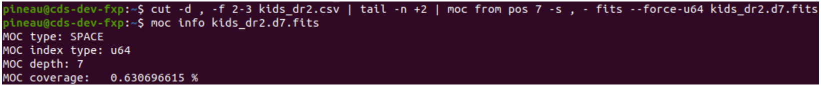
□ MOCCli examples

- Create an allsky MOC at order 8 in FITS format

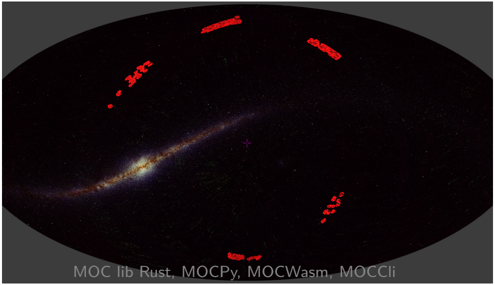


```
pineau@cds-dev-fxp:~$ echo "0/0-11 8/" | moc convert --type smoc --format ascii - fits moc.d8.allsky.fits
pineau@cds-dev-fxp:~$ moc info moc.d8.allsky.fits
MOC type: SPACE
MOC index type: u32
MOC depth: 8
MOC coverage: 100.000000000 %
```

- Create an MOC from positions in a CSV file



```
pineau@cds-dev-fxp:~$ cut -d , -f 2-3 kids_dr2.csv | tail -n +2 | moc from pos 7 -s , - fits --force-u64 kids_dr2.d7.fits
pineau@cds-dev-fxp:~$ moc info kids_dr2.d7.fits
MOC type: SPACE
MOC index type: u64
MOC depth: 7
MOC coverage:   0.630696615 %
```



MOCCli examples

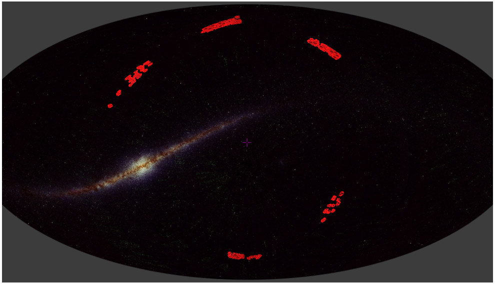
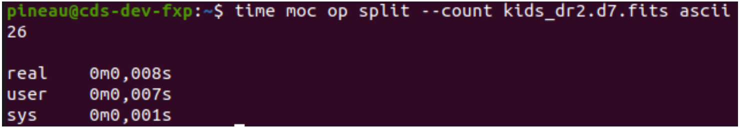


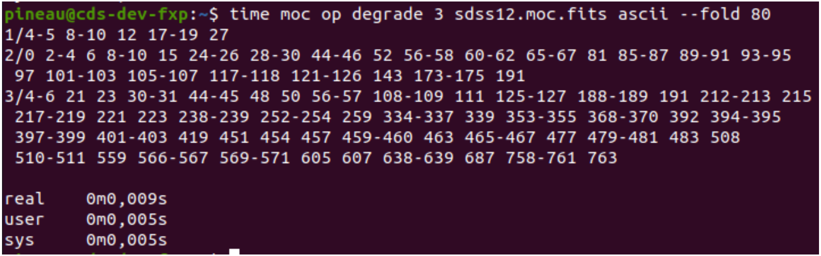
Figure 3: KIDS DR2 MOC at order 7



```
pineau@cds-dev-fxp:~$ time moc op split --count kids_dr2.d7.fits ascii
26

real	0m0,008s
user	0m0,007s
sys	0m0,001s
```

Figure 4: Count the number of disjoint MOCs

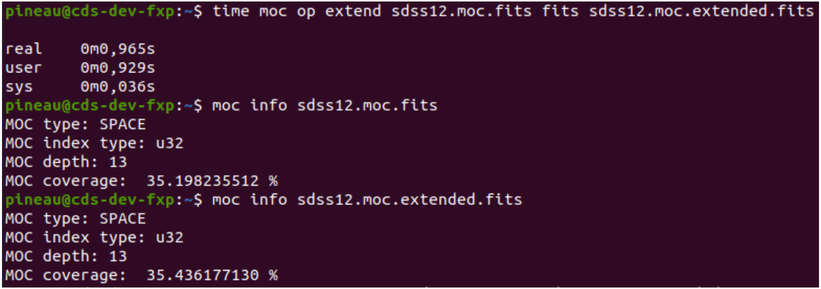
MOCCli examples



```
pineau@cds-dev-fxp:~$ time moc op degrade 3 sdss12.moc.fits ascii --fold 80
1/4-5 8-10 12 17-19 27
2/0 2-4 6 8-10 15 24-26 28-30 44-46 52 56-58 60-62 65-67 81 85-87 89-91 93-95
 97 101-103 105-107 117-118 121-126 143 173-175 191
3/4-6 21 23 30-31 44-45 48 50 56-57 108-109 111 125-127 188-189 191 212-213 215
 217-219 221 223 238-239 252-254 259 334-337 339 353-355 368-370 392 394-395
 397-399 401-403 419 451 454 457 459-460 463 465-467 477 479-481 483 508
 510-511 559 566-567 569-571 605 607 638-639 687 758-761 763

real	0m0,009s
user	0m0,005s
sys	0m0,005s
```

Figure 5: Degrade the SDSS DR12 MOC at order 3 and print the result in ASCII in stdout

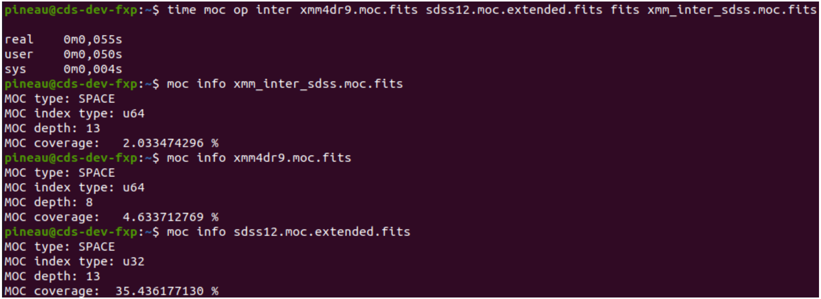
MOCCli examples



```
pineau@cds-dev-fxp:~$ time moc op extend sdss12.moc.fits fits sdss12.moc.extended.fits

real	0m0,965s
user	0m0,929s
sys	0m0,036s
pineau@cds-dev-fxp:~$ moc info sdss12.moc.fits
MOC type: SPACE
MOC index type: u32
MOC depth: 13
MOC coverage:  35.198235512 %
pineau@cds-dev-fxp:~$ moc info sdss12.moc.extended.fits
MOC type: SPACE
MOC index type: u32
MOC depth: 13
MOC coverage:  35.436177130 %
```

Figure 6: Extend the SDSS DR12 MOC with a border of depth 13 cells

MOCCli examples



```
pineau@cds-dev-fxp:~$ time moc op inter xmm4dr9.moc.fits sdss12.moc.extended.fits fits xmm_inter_sdss.moc.fits

real    0m0,055s
user    0m0,050s
sys     0m0,004s
pineau@cds-dev-fxp:~$ moc info xmm_inter_sdss.moc.fits
MOC type: SPACE
MOC index type: u64
MOC depth: 13
MOC coverage:   2.033474296 %
pineau@cds-dev-fxp:~$ moc info xmm4dr9.moc.fits
MOC type: SPACE
MOC index type: u64
MOC depth: 8
MOC coverage:   4.633712769 %
pineau@cds-dev-fxp:~$ moc info sdss12.moc.extended.fits
MOC type: SPACE
MOC index type: u32
MOC depth: 13
MOC coverage:  35.436177130 %
```

Figure 7: Compute the intersection between the XMM MOC and the SDSS DR12 extended MOC

MOCWasm

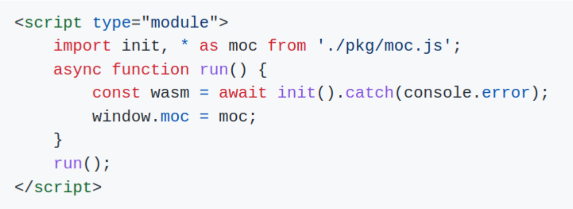
- Manipulate MOCs in Web Browsers:
  - JavaScript (JS) API calling WebAssembly (WASM) code
- Open source
  - https://github.com/cds-astro/cds-moc-rust/tree/main/crates/wasm
  - Apache 2.0 + MIT dual licensing
- See https://github.com/cds-astro/cds-moc-rust/releases
  - single package containing 2 files
    - `moc.js` (63 KB)
    - `moc_bg.wasm` (664 KB)
- Based on wasm-bindgen
  - Pure Rust: not a single line of JavaScript!

Use MOCWasm

- Download and extract a MOCWasm release

```
> wget https://github.com/cds-astro/cds-moc-rust/releases
/download/v0.4.0/moc-wasm-v0.4.0.tar.gz
> tar xvzf moc-wasm-v0.4.0.tar.gz
pkg/
pkg/moc_bg.wasm
pkg/moc.js
...
```

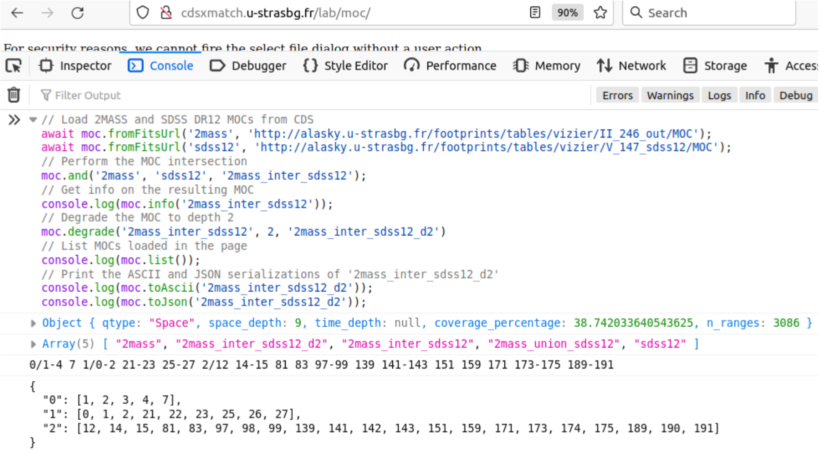
- Add it to your web page HTML body



```
<script type="module">
    import init, * as moc from './pkg/moc.js';
    async function run() {
        const wasm = await init().catch(console.error);
        window.moc = moc;
    }
    run();
</script>
```

MOCWasm demo page

- Basic page to use MOCWasm from the Web Browser console: http://cdsxmatch.u-strasbg.fr/lab/moc/

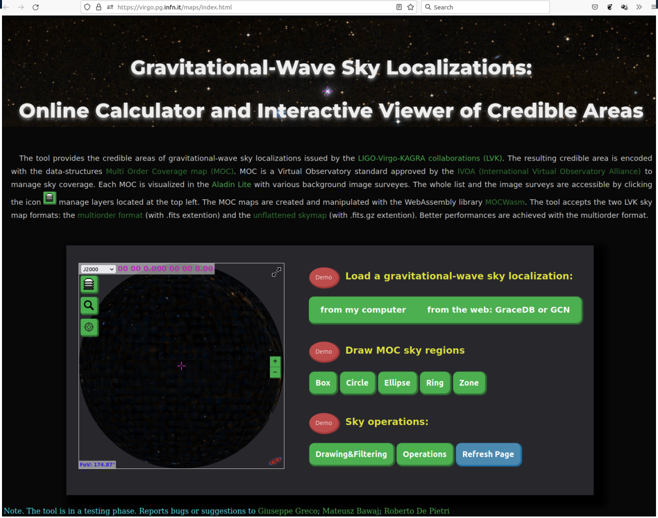


For security reasons, we cannot fire the select file dialog without a user action

```
// Load 2MASS and SDSS DR12 MOCs from CDS
await moc.fromFitsUrl('2mass', 'http://alasky.u-strasbg.fr/footprints/tables/vizier/II_246_out/MOC');
await moc.fromFitsUrl('sdss12', 'http://alasky.u-strasbg.fr/footprints/tables/vizier/V_147_sdss12/MOC');
// Perform the MOC intersection
moc.and('2mass', 'sdss12', '2mass_inter_sdss12');
// Get info on the resulting MOC
console.log(moc.info('2mass_inter_sdss12'));
// Degrade the MOC to depth 2
moc.degrade('2mass_inter_sdss12', 2, '2mass_inter_sdss12_d2')
// List MOCs loaded in the page
console.log(moc.list());
// Print the ASCII and JSON serializations of '2mass_inter_sdss12_d2'
console.log(moc.toAscii('2mass_inter_sdss12_d2'));
console.log(moc.toJson('2mass_inter_sdss12_d2'));
```

```
Object { qtype: "Space", space_depth: 9, time_depth: null, coverage_percentage: 38.742033640543625, n_ranges: 3086 }
Array(5) [ "2mass", "2mass_inter_sdss12_d2", "2mass_inter_sdss12", "2mass_union_sdss12", "sdss12" ]
0/1-4 7 1/0-2 21-23 25-27 2/12 14-15 81 83 97-99 139 141-143 151 159 171 173-175 189-191
{
  "0": [1, 2, 3, 4, 7],
  "1": [0, 1, 2, 21, 22, 23, 25, 26, 27],
  "2": [12, 14, 15, 81, 83, 97, 98, 99, 139, 141, 142, 143, 151, 159, 171, 173, 174, 175, 189, 190, 191]
}
```

MOCWasm example



https://virgo.pg.infn.it/maps/index.html

Gravitational-Wave Sky Localizations:

Online Calculator and Interactive Viewer of Credible Areas

The tool provides the credible areas of gravitational-wave sky localizations issued by the LIGO-Virgo-KAGRA collaborations (LVK). The resulting credible area is encoded with the data-structures Multi Order Coverage map (MOC). MOC is a Virtual Observatory standard approved by the IVOA (International Virtual Observatory Alliance) to manage sky coverage. Each MOC is visualized in the Aladin Lite with various background image surveyes. The whole list and the image surveys are accessible by clicking the icon 🟩 manage layers located at the top left. The MOC maps are created and manipulated with the WebAssembly library MOCWasm. The tool accepts the two LVK sky map formats: the multiorder format (with .fits extention) and the unflattened skymap (with .fits.gz extention). Better performances are achieved with the multiorder format.

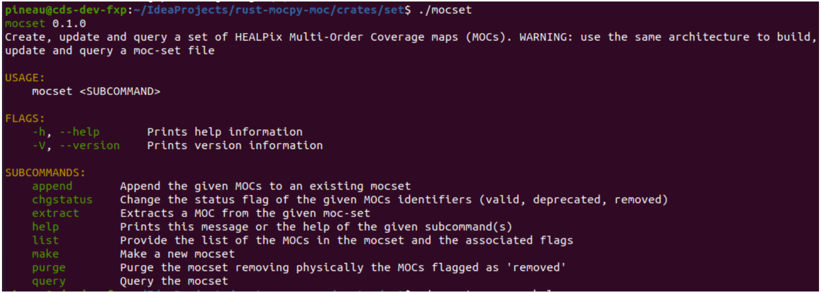
Note. The tool is in a testing phase. Reports bugs or suggestions to Giuseppe Greco; Mateusz Bawaj; Roberto De Pietri

Figure 8: Web page https://virgo.pg.infn.it/maps/index.html by Giuseppe Greco; Mateusz Bawaj; Roberto De Pietri

□ MOCSet

- A command line tool to build, update and query a set of MOCs
- A kind of simplified MOCserver
  - See the Moc Server presentation by P.Fernique
  - but with a different architecture
- Use case:
  - VizieR (in testing phase by G. Landais)
    - list tables intersected by a cone
  - SSC XMM: list observations contanING a given polygon
- Performances
  - tested with 25,287 VizieR MOCs at various orders (mainly 11)
  - binary file of 3.1 GB
  - cold cache (HDD): query in about 16 s
  - cold cache (NVMe SSD): query in less than 1.5 s
  - hot cache: query in about 60 ms

MOCSet CLI



```
pineau@cds-dev-fxp:~/IdeaProjects/rust-mocpy-moc/crates/set$ ./mocset
mocset 0.1.0
Create, update and query a set of HEALPix Multi-Order Coverage maps (MOCs). WARNING: use the same architecture to build,
update and query a moc-set file

USAGE:
    mocset <SUBCOMMAND>

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

SUBCOMMANDS:
    append       Append the given MOCs to an existing mocset
    chgstatus    Change the status flag of the given MOCs identifiers (valid, deprecated, removed)
    extract      Extracts a MOC from the given moc-set
    help         Prints this message or the help of the given subcommand(s)
    list         Provide the list of the MOCs in the mocset and the associated flags
    make         Make a new mocset
    purge        Purge the mocset removing physically the MOCs flagged as 'removed'
    query        Query the mocset
```

MOCSet: technically

- Uses memory maps
  - memory footprint managed by the OS
  - => possibly low memory consumption
  - => but performances depends on cache size (and HDD vs SSD)
- Directly maps MOC structures on memory mapped bytes
  - no pre-loading, no copy
  - => super fast (like old-style C)
- Designed to be updatable while reading, without locking
  - super fast updates (except `purge`)

Conclusion

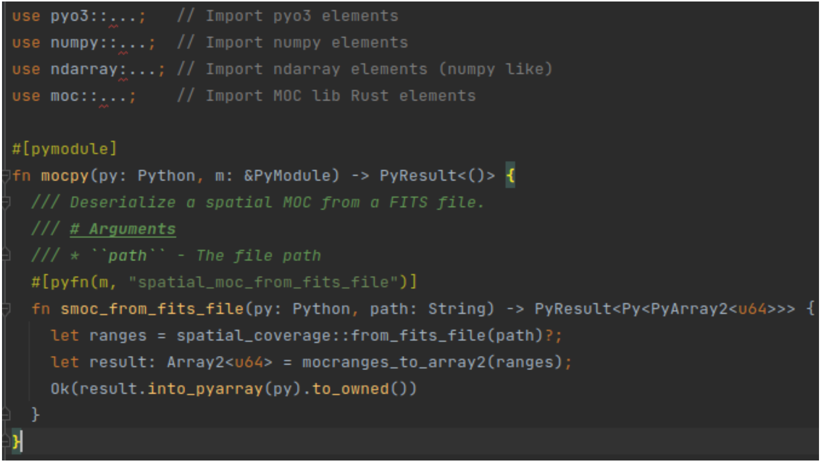
- A single language (Rust), then use wrappers:
  - PyO3 for Python
  - Wasm-bindgen for Javascript/WebAssembly
  - StructOpt to build command line tools
- Future
  - distribute MOCCli through pypi?
  - distribute MOCWasm through npm?
  - C/PostgreSQL wrapper (using bindgen)?
  - improve ST-MOC support
  - add tests, improve doc, ...
- URLs:
  - MOC lib Rust: `https://github.com/cds-astro/cds-moc-rust`
  - MOCPy: `https://github.com/cds-astro/mocpy`
  - MOCCli and MOCWasm last release: `https://github.com/cds-astro/cds-moc-rust/releases/tag/v0.4.0`

Why Rust? (biased)

- No C/C++ expert background
- Pro
  - Fast and safe (no segmentation fault)
    - both low and high level, zero-cost abstractions (like C++)
    - "Once it compiles, it works" (like Java)
  - Multi-paradigm, compiled, no runtime, no GC
  - 2nd Linux kernel language; used in Dropbox, Firefox, npm, . . .
  - Modern tooling (Cargo)
  - Targets WebAssembly, see wasm-bindgen
  - Easy binding with C, see bindgen
    - Expose Rust libs as a C libs (and vice-versa)
  - Supposedly a green language
  - Personal taste: the language I have always looked for
- Con
  - Chicken and egg problem (limited libs/adoption inertia)
  - Steep learning curve: see how-not-to-learn-rust

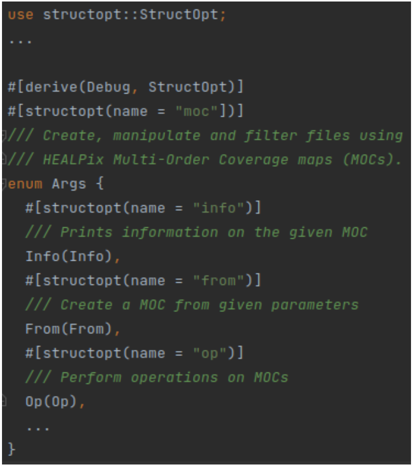
MOCpy: PyO3 binding

Example from the MOCPy Rust source code:



```
use pyo3::...;    // Import pyo3 elements
use numpy::...;   // Import numpy elements
use ndarray:...; // Import ndarray elements (numpy like)
use moc::...;     // Import MOC lib Rust elements

#[pymodule]
fn mocpy(py: Python, m: &PyModule) -> PyResult<()> {
  /// Deserialize a spatial MOC from a FITS file.
  /// # Arguments
  /// * ``path`` - The file path
  #[pyfn(m, "spatial_moc_from_fits_file")]
  fn smoc_from_fits_file(py: Python, path: String) -> PyResult<Py<PyArray2<u64>>> {
    let ranges = spatial_coverage::from_fits_file(path)?;
    let result: Array2<u64> = mocranges_to_array2(ranges);
    Ok(result.into_pyarray(py).to_owned())
  }
}
```

□ MOCCli: Structopt

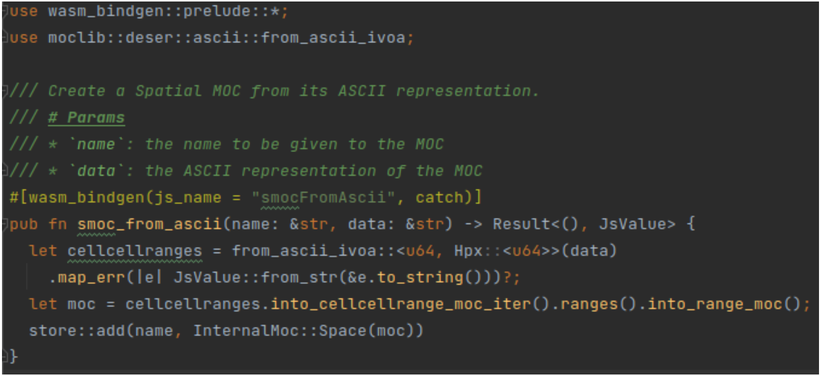


```
use structopt::StructOpt;
...

#[derive(Debug, StructOpt)]
#[structopt(name = "moc")]
/// Create, manipulate and filter files using
/// HEALPix Multi-Order Coverage maps (MOCs).
enum Args {
  #[structopt(name = "info")]
  /// Prints information on the given MOC
  Info(Info),
  #[structopt(name = "from")]
  /// Create a MOC from given parameters
  From(From),
  #[structopt(name = "op")]
  /// Perform operations on MOCs
  Op(Op),
  ...
}
```

Figure 9: Example from the MOCCli source code using StructOpt

MOCWasm internals

- Rust – JS/WASM binding made by wasm-bindgen



```
use wasm_bindgen::prelude::*;
use moclib::deser::ascii::from_ascii_ivoa;

/// Create a Spatial MOC from its ASCII representation.
/// # Params
/// * `name`: the name to be given to the MOC
/// * `data`: the ASCII representation of the MOC
#[wasm_bindgen(js_name = "smocFromAscii", catch)]
pub fn smoc_from_ascii(name: &str, data: &str) -> Result<(), JsValue> {
  let cellcellranges = from_ascii_ivoa::<u64, Hpx::<u64>>(data)
    .map_err(|e| JsValue::from_str(&e.to_string()))?;
  let moc = cellcellranges.into_cellcellrange_moc_iter().ranges().into_range_moc();
  store::add(name, InternalMoc::Space(moc))
}
```

- All the magic is in the #[wasm_bindgen()] macro
  - boilerplate code automatically generated