Module Anatomy

This page explains all of the elements needed to develop and plug-in a new MADlib® module.

1. Module files overview (source tree perspective)
2. Module files explained
3. Configuration
4. Adding support for other DB platforms

1. Module files overview (source tree perspective)

Say you want to write a new MADlib module called `NewModule` (code name: `newmod`). Use the following directory tree structure as a reference for your module:

```
./src/
    modules/
        newmod/             # (optional) new directory for the module code
            newmod.cpp      # (optional) C/C++ code for this module
            newmod.hpp      # (optional) C/C++ header for this module
            ...
    ports/
        postgres/
            modules/
                newmod/
                    newmod.sql_in    # (REQUIRED) SQL file to create DB objects
                    newmod.py_in     # (optional) Python code (helpful for iterative algorithms)
                    test/            # (optional) directory for SQL test scripts
                        newmod.sql_in    # (optional) test scripts that will be run during install-check
                    ...
```

2. Module files explained

- **newmod.sql_in** - SQL file which creates database objects for this method. This is the only required code file, because there could me a module /method written completely in SQL. There would be no need for Python or C/C++ code in such case. This file is pre-processed with m4 during installation phase and currently uses the following meta variables:
    - MADLIB_SCHEMA - will be replaced with the target schema name
    - PLPYTHON_LIBDIR - used inside PL/Python routines (UDFs) and will be replaced with a path to a directory with the Python module of each method
    - MODULE_PATHNAME - used inside C routines (UDFs) and will be replaced with a path to a directory with the C/C++ module of each method
- **newmod.py_in** - Python code for `newmod` module. A Python layer helps in pre/post-processing data before the module logic kicks in and is also useful in running iterative algorithms. This logic can be implemented in the PostgreSQL procedural language, but is usually simplified in Python.
- **newmod.c/cpp** - C/C++ code for `newmod` module. This is the logic that is executed in each iteration. Implementing in C++ leads to performant code when compared with implementing in SQL or PL/pgSQL.
- **test/newmod.sql_in** - SQL test script that is executed during install-check

3. Configuration:

In order to include the new module in the generic (not database dependent) installation, only the following config file must be edited: `./config/Modules.yml`. New `name` element must be added with an optional `depends` item:

```
- name:    newmod
  depends: ['othermod1', 'othermod2']
```

4. Adding support for other DB platforms:

If you must adjust any of the code to a particular database platform the files which requires changes must be replicated under a dedicated `./port/<portid>/module` directory, see below. The database `<portid>` you will be referring to must be already defined in `./config/Ports.yml` config file.

```
./ports/
    greenplum/                  # Example port id: greenplum
        modules/
            newmod/             # (REQUIRED) new directory for the module code
                newmod.sql_in   # (optional) SQL file to create DB objects
                newmod.py_in    # (optional) Python code
                ...
```