Compatibility Between ESDM and NetCDF4

Luciana Pedro Julian Kunkel Benjamin Hodges

Work Package: Work Package 4 Exploitability

Date: March 4, 2020
History October 2019, Version 0.9.
January 2020, Integrated behavior of the current EDSM development

Contents

1. Python Tests

1.1. Introduction

There is a Python module for accessing NetCDF files. We provide a patch to the module to support ESDM. The patch and instructions to run the tests are available in the directory **esdm-netcdf/dev**. The procedures to run the tests are following.

Run the build script:

```
$ ./build.sh
```

Python tests are created now inside the directory **dev/netcdf4-python/test**. Go to that directory.

```
$ cd netcdf4-python/test
```

Now, copy or link the **_esdm.conf** file to this directory. A possible example is:

```
$ cp ../../../build/libsrcesdm_test/_esdm.conf .
```

All Python tests start with the prefix **tst_**. This string has to change to enable us to run **pytest**. To do that, one option is use the the **mmv** utility. Note that the tests names need to be changed, but the tests files (**.nc** files) have to remain unchanged.

The tool can be installed and used in Debian-based distributions as follows:

```
$ sudo apt-get install mmv
```

Now, run the **mmv** in the tests.

```
$ mmv tst.py test#1.py
```

and run the **mkfs.esdm** utility with the following parameters.

```
$ mkfs.esdm –create –remove –ignore-errors -g -c _esdm.conf
```

Finally, install the **pytest** utility

```
$ sudo apt-get install python3-pytest
```

and run it

```
$ pytest –junitxml results.xml
```

The file **results.xml** can now be uploaded by Jenkins wchi provides the results in Table tab.

1.2. Patch

```
diff --git a/include/netCDF4.pxi b/include/netCDF4.pxi
index 625752a..7bcd019 100644
--- a/include/netCDF4.pxi
+++ b/include/netCDF4.pxi
@@ -49,6 +49,7 @@ cdef extern from "netcdf.h":
         NC_WRITE # read & write
         # Use these 'mode' flags for nc_create.
         NC_CLOBBER
+        NC_ESDM # ESDM support for Python
         NC_NOCLOBBER # Don't destroy existing file on create
         NC_64BIT_OFFSET # Use large (64-bit) file offsets
         NC_64BIT_DATA # Use cdf-5 format
@@ -122,6 +123,7 @@ cdef extern from "netcdf.h":
         NC_FORMAT_NC3
         NC_FORMAT_NC_HDF4
         NC_FORMAT_NC_HDF5
+        NC_FORMATX_ESDM
         NC_FORMAT_DAP2
         NC_FORMAT_DAP4
         NC_FORMAT_PNETCDF
@@ -704,7 +706,7 @@ IF HAS_NC_CREATE_MEM:
             int flags
         int nc_close_memio(int ncid, NC_memio* info);

-IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
     cdef extern from "mpi-compat.h": pass
     cdef extern from "netcdf_par.h":
         ctypedef int MPI_Comm
diff --git a/netCDF4/_netCDF4.pyx b/netCDF4/_netCDF4.pyx
index b693354..58d02e8 100644
--- a/netCDF4/_netCDF4.pyx
+++ b/netCDF4/_netCDF4.pyx
@@ -1264,7 +1264,7 @@ import_array()
 include "constants.pyx"
 include "membuf.pyx"
 include "netCDF4.pxi"
-IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
     cimport mpi4py.MPI as MPI
     from mpi4py.libmpi cimport MPI_Comm, MPI_Info, MPI_Comm_dup, MPI_Info_dup, \
                                MPI_Comm_free, MPI_Info_free, MPI_INFO_NULL ,\
@@ -1344,11 +1344,13 @@ _intnptonctype  = {'i1' : NC_BYTE,
 # create dictionary mapping string identifiers to netcdf format codes
 _format_dict  = {'NETCDF3_CLASSIC' : NC_FORMAT_CLASSIC,
                  'NETCDF4_CLASSIC' : NC_FORMAT_NETCDF4_CLASSIC,
-                 'NETCDF4'         : NC_FORMAT_NETCDF4}
+                 'NETCDF4'         : NC_FORMAT_NETCDF4,
+                 'ESDM'            : NC_FORMATX_ESDM}
 # create dictionary mapping string identifiers to netcdf create format codes
 _cmode_dict  = {'NETCDF3_CLASSIC' : NC_CLASSIC_MODEL,
                 'NETCDF4_CLASSIC' : NC_CLASSIC_MODEL | NC_NETCDF4,
-                'NETCDF4'         : NC_NETCDF4}
+                'NETCDF4'         : NC_NETCDF4,
+                'ESDM'            : NC_ESDM}
 IF HAS_CDF5_FORMAT:
     # NETCDF3_64BIT deprecated, saved for compatibility.
     # use NETCDF3_64BIT_OFFSET instead.
@@ -1547,6 +1549,8 @@ cdef _get_full_format(int grpid):
             return 'DAP2'
         elif formatp == NC_FORMAT_DAP4:
             return 'DAP4'
+        elif formatp == NC_FORMATX_ESDM:
+            return 'ESDM'
         elif formatp == NC_FORMAT_UNDEFINED:
             return 'UNDEFINED'
     ELSE:
@@ -1578,7 +1582,7 @@ be raised in the next release."""
                 PyMem_Free(string_ptrs)
         else:
             # don't allow string array attributes in NETCDF3 files.
-            if is_netcdf3 and N > 1:
+            if is_netcdf3 and N > 1 and fmt != 'ESDM':
                 msg='array string attributes can only be written with NETCDF4'
                 raise IOError(msg)
             if not value_arr.shape:
@@ -1593,7 +1597,7 @@ be raised in the next release."""
     # if array is 64 bit integers or
     # if 64-bit datatype not supported, cast to 32 bit integers.
     fmt = _get_format(grp._grpid)
-    is_netcdf3 = fmt.startswith('NETCDF3') or fmt == 'NETCDF4_CLASSIC'
+    is_netcdf3 = fmt.startswith('NETCDF3') or fmt == 'NETCDF4_CLASSIC' or fmt == 'ESDM'
     if value_arr.dtype.str[1:] == 'i8' and ('i8' not in _supportedtypes or \
        (is_netcdf3 and fmt != 'NETCDF3_64BIT_DATA')):
         value_arr = value_arr.astype('i4')
@@ -2030,7 +2034,7 @@ strings.
     __pdoc__['Dataset.data_model']=\
     """`data_model` describes the netCDF
     data model version, one of `NETCDF3_CLASSIC`, `NETCDF4`,
-    `NETCDF4_CLASSIC`, `NETCDF3_64BIT_OFFSET` or `NETCDF3_64BIT_DATA`."""
+    `NETCDF4_CLASSIC`, `NETCDF3_64BIT_OFFSET` or `NETCDF3_64BIT_DATA` or ESDM."""
     __pdoc__['Dataset.file_format']=\
     """same as `data_model`, retained for backwards compatibility."""
     __pdoc__['Dataset.disk_format']=\
@@ -2084,7 +2088,7 @@ strings.
 
         **`format`**: underlying file format (one of `'NETCDF4',
         'NETCDF4_CLASSIC', 'NETCDF3_CLASSIC'`, `'NETCDF3_64BIT_OFFSET'` or
-        `'NETCDF3_64BIT_DATA'`.
+        `'NETCDF3_64BIT_DATA'` or ESDM.
         Only relevant if `mode = 'w'` (if `mode = 'r','a'` or `'r+'` the file format
         is automatically detected). Default `'NETCDF4'`, which means the data is
         stored in an HDF5 file, using netCDF 4 API features.  Setting
@@ -2156,7 +2160,7 @@ strings.
         cdef char *path
         cdef char namstring[NC_MAX_NAME+1]
         cdef int cmode
-        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
             cdef MPI_Comm mpicomm
             cdef MPI_Info mpiinfo

@@ -2183,7 +2187,7 @@ strings.
                 msg='parallel mode requires MPI enabled netcdf-c'
                 raise ValueError(msg)
             ELSE:
-                parallel_formats = []
+                parallel_formats = ['ESDM']
                 IF HAS_PARALLEL4_SUPPORT:
                     parallel_formats += ['NETCDF4','NETCDF4_CLASSIC']
                 IF HAS_PNETCDF_SUPPORT:
@@ -2222,7 +2226,7 @@ strings.
             else:
                 if clobber:
                     if parallel:
-                        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+                        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
    ↪  HAS_ESDM_SUPPORT:
                             ierr = nc_create_par(path, NC_CLOBBER | cmode,
    ↪  \
                                    mpicomm, mpiinfo, &grpid)
                         ELSE:
@@ -2272,7 +2276,7 @@ strings.
         version 4.4.1 or higher of the netcdf C lib, and rebuild netcdf4-
    ↪ python."""
                     raise ValueError(msg)
             elif parallel:
                 IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+                IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
    ↪ HAS_ESDM_SUPPORT:
                     ierr = nc_open_par(path, NC_NOWRITE | NC_MPIIO, \
                            mpicomm, mpiinfo, &grpid)
                 ELSE:
@@ -2853,7 +2857,7 @@ Use if you need to ensure that a netCDF attribute is
    ↪ created with type
 `NC_STRING` if the file format is `NETCDF4`."""
         cdef nc_type xtype
         xtype=-99
-        if self.data_model != 'NETCDF4':
+        if self.data_model != 'NETCDF4' and self.data_model != 'ESDM':
             msg='file format does not support NC_STRING attributes'
             raise IOError(msg)
         _set_att(self, NC_GLOBAL, name, value, xtype=xtype, force_ncstring
    ↪ =True)
@@ -3435,7 +3435,7 @@ return the group that this `netCDF4.Dimension` is a
    ↪ member of."""
 returns `True` if the `netCDF4.Dimension` instance is unlimited, `False`
    ↪ otherwise."""
         cdef int ierr, n, numunlimdims, ndims, nvars, ngatts, xdimid
         cdef int *unlimdimids
-        if self._data_model == 'NETCDF4':
+        if self._data_model == 'NETCDF4' or self._data_model == "ESDM":
             ierr = nc_inq_unlimdims(self._grpid, &numunlimdims, NULL)
             _ensure_nc_success(ierr)
             if numunlimdims == 0:
@@ -4138,7 +4138,7 @@ Use if you need to ensure that a netCDF attribute is
    ↪ created with type
 Use if you need to set an attribute to an array of variable-length strings
    ↪ ."""
         cdef nc_type xtype
         xtype=-99
-        if self._grp.data_model != 'NETCDF4':
+        if self._grp.data_model != 'NETCDF4' and self._grp.data_model != '
    ↪ ESDM':
             msg='file format does not support NC_STRING attributes'
             raise IOError(msg)
         _set_att(self._grp, self._varid, name, value, xtype=xtype,
    ↪ force_ncstring=True)
diff --git a/setup.py b/setup.py
index febc020..d7ba091 100644
--- a/setup.py
+++ b/setup.py
@@ -58,6 +58,7 @@ def check_api(inc_dirs):
     has_nc_create_mem = False
     has_parallel4_support = False
     has_pnetcdf_support = False
+    has_esdm_support = False

     for d in inc_dirs:
         try:
@@ -564,6 +565,7 @@ if 'sdist' not in sys.argv[1:] and 'clean' not in sys.
    ↪ argv[1:]:
     else:
         sys.stdout.write('netcdf lib does not have pnetcdf parallel
    ↪ functions\n')
         f.write('DEF HAS_PNETCDF_SUPPORT = 0\n')
+    f.write('DEF HAS_ESDM_SUPPORT = 1\n') # TODO Fixme

     f.close()
```

```
+        NC_ESDM # ESDM support for Python
+        NC_FORMATX_ESDM
-IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
-IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
-                     'NETCDF4'          : NC_FORMAT_NETCDF4}
+                     'NETCDF4'          : NC_FORMAT_NETCDF4,
+                     'ESDM'             : NC_FORMATX_ESDM}
-                    'NETCDF4'          : NC_NETCDF4}
+                    'NETCDF4'          : NC_NETCDF4,
+                    'ESDM'             : NC_ESDM}
+        elif formatp == NC_FORMATX_ESDM:
+            return 'ESDM'
-            if is_netcdf3 and N > 1:
+            if is_netcdf3 and N > 1 and fmt != 'ESDM':
-    is_netcdf3 = fmt.startswith('NETCDF3') or fmt == 'NETCDF4_CLASSIC'
+    is_netcdf3 = fmt.startswith('NETCDF3') or fmt == 'NETCDF4_CLASSIC' or
  ↪ fmt == 'ESDM'
-    `NETCDF4_CLASSIC`, `NETCDF3_64BIT_OFFSET` or `NETCDF3_64BIT_DATA`."""
+    `NETCDF4_CLASSIC`, `NETCDF3_64BIT_OFFSET` or `NETCDF3_64BIT_DATA` or
  ↪ ESDM."""
-        `'NETCDF3_64BIT_DATA'`.
+        `'NETCDF3_64BIT_DATA'` or ESDM.
-        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
  ↪ HAS_ESDM_SUPPORT:
-                parallel_formats = []
+                parallel_formats = ['ESDM']
-                        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+                        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
  ↪  HAS_ESDM_SUPPORT:
-                IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+                IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
  ↪ HAS_ESDM_SUPPORT:
-        if self.data_model != 'NETCDF4':
+        if self.data_model != 'NETCDF4' and self.data_model != 'ESDM':
-        if self._data_model == 'NETCDF4':
+        if self._data_model == 'NETCDF4' or self._data_model == "ESDM":
-        if self._grp.data_model != 'NETCDF4':
+        if self._grp.data_model != 'NETCDF4' and self._grp.data_model != '
  ↪ ESDM':
+    has_esdm_support = False
+    f.write('DEF HAS_ESDM_SUPPORT = 1\n') # TODO Fixme
```

```
+        NC_ESDM # ESDM support for Python
+        NC_FORMATX_ESDM
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
+                     'NETCDF4'          : NC_FORMAT_NETCDF4,
+                     'ESDM'             : NC_FORMATX_ESDM}
+                    'NETCDF4'          : NC_NETCDF4,
+                    'ESDM'             : NC_ESDM}
+        elif formatp == NC_FORMATX_ESDM:
+            return 'ESDM'
+            if is_netcdf3 and N > 1 and fmt != 'ESDM':
+    is_netcdf3 = fmt.startswith('NETCDF3') or fmt == 'NETCDF4_CLASSIC' or
  ↪ fmt == 'ESDM'
+    `NETCDF4_CLASSIC`, `NETCDF3_64BIT_OFFSET` or `NETCDF3_64BIT_DATA` or
  ↪ ESDM."""
+        `'NETCDF3_64BIT_DATA'` or ESDM.
+        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
  ↪ HAS_ESDM_SUPPORT:
+                parallel_formats = ['ESDM']
+                        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
  ↪  HAS_ESDM_SUPPORT:
+                IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
  ↪ HAS_ESDM_SUPPORT:
+        if self.data_model != 'NETCDF4' and self.data_model != 'ESDM':
+        if self._data_model == 'NETCDF4' or self._data_model == "ESDM":
+        if self._grp.data_model != 'NETCDF4' and self._grp.data_model != '
  ↪ ESDM':
+    has_esdm_support = False
+    f.write('DEF HAS_ESDM_SUPPORT = 1\n')
```

2. ESDM Implementation

2.1. Introduction

This chapter describes how ESDM objects are related to NetCDF objects. For each NetCDF file, ESDM creates a container, and for each NetCDF variable, ESDM creates a dataset. To be able to interact with the NetCDF C code, we introduce the file **esdm_dispatch.c**. This file contains the functions that will be used by ESDM that will be called by the dispatch table when NetCDF runs. For each `nc_function`, we have an `ESDM_function` with the same parameters, providing the same outcome, but using this new middleware. Some of the ESDM functions use the flag ESDM_PARALLEL to check if the code is being parallelised and call the respective ESDM parallel functions if that is the case.

2.1.1. ESDM Open

ESDM is first called when `nc_open` is called, which triggers the respective function `ESDM_open` and loads the data from the NetCDF file to memory. Then, the following functions are called to create a representation of the NetCDF file according to the ESDM entities.

esdm_container_open Open the container that contains the NetCDF file.

esdm_container_dataset_count Check the number of datasets in the container.

esdm_container_dataset_from_array For each dataset, load it into memory.

esdm_dataset_ref For each dataset, check whether the dataset was already loaded into memory. If that is not the case, the dataset's metadata is loaded into memory before incrementing the reference count.

esdm_dataset_get_dataspace For each dataset, create a copy of the internal dataspace object.

esdm_dataspace_get_dims For each dataset, read the number of dimensions from the dataspace.

esdm_dataset_get_name_dims For each dimension, read its name.

esdm_dataset_get_size For each dimension, read its size.

add_to_dims_tbl For each dataset and for each dimension, insert the dimension into the dimension table, if not already there. The dimension table is constructed in memory to allocate the number of dimensions, sizes, and names.

insert_md For each dataset, insert the information about its metadata.

esdm_container_get_attributes Open the global attributes of the NetCDF file.

add_to_dims_tbl If there exists a dimension that was not previously used in any dataset, insert this dimension into the dimension table.

2.1.2. ESDM Close

When `nc_close` is called, the respective function `ESDM_close` is triggerred to close the NetCDF file. T following functions are called to close the NetCDF file according to the ESDM entities.

ESDM_nc_get_esdm_struct Loads the data from the NetCDF file to memory.

ncesdm_container_commit Insert the information from the dimension table as a global attribute and make the information inside the container persistent.

esdm_container_dataset_count Check the number of datasets in the container.

esdm_container_dataset_from_array For each dataset, load it into memory.

esdm_dataset_close For each dataset, remove its information from storage.

esdm_container_close Remove the container from storage.

Listing 2.1: ESDM Entities

```c

struct esdm_datasets_t {
  esdm_dataset_t ** dset;
  int count;
  int buff_size;
};

struct esdm_container_t {
  char *name;
  smd_attr_t *attr;
  esdm_datasets_t dsets;

  int refcount;
  esdm_data_status_e status;
  int mode_flags; // set via esdm_mode_flags_e
};

typedef struct esdmI_hypercubeNeighbourManager_t
    ↪ esdmI_hypercubeNeighbourManager_t;
struct esdm_fragments_t {
  esdm_fragment_t ** frag;
  esdmI_hypercubeNeighbourManager_t* neighbourManager;
  int count;
  int buff_size;
};

typedef struct esdm_fragments_t esdm_fragments_t;

struct esdm_dataset_t {
  char *name;
  char *id;
  char **dims_dset_id; // array of variable names != NULL if set
  esdm_container_t *container;
  esdm_dataspace_t *dataspace;
  smd_attr_t *fill_value; // use for read of not-written data, if set
  smd_attr_t *attr;
  int64_t *actual_size; // used for unlimited dimensions
  esdm_fragments_t fragments;
  int refcount;
  esdm_data_status_e status;
  int mode_flags; // set via esdm_mode_flags_e
};

struct esdm_fragment_t {
  char * id;
  esdm_dataset_t *dataset;
  esdm_dataspace_t *dataspace;
  esdm_backend_t *backend;
  void * backend_md; // backend-specific metadata if set
  void *buf;
  size_t elements;
  size_t bytes;
  //int direct_io;
  esdm_data_status_e status;
};

struct esdm_dataspace_t {
  esdm_type_t type;
  int64_t dims;
  int64_t *size;
  int64_t *offset;
  int64_t *stride;  //may be NULL, in this case contiguous storage in C
    ↪ order is assumed
};
```

Listing 2.2: Structures kept in memory by ESDM

```c
typedef struct {
  int *dimidsp;
  int fillmode; // remembers if fill mode is on or off, by default fill
    ↪ mode is
                // on, actually, ESDM with NetCDF always has a fill mode
    ↪ , uses
                // the defaults from NetCDF
  esdm_dataset_t *dset;
} md_var_t;

typedef struct {
  int count;
  md_var_t **var;
} md_vars_t;

typedef struct {
  int count;
  uint64_t *size;
  char **name;
} nc_dim_tbl_t;

typedef struct {
  int ncid;
  int fillmode; // remembers if fill mode is on or off, by default fill
    ↪ mode is
                // on, actually, ESDM with NetCDF always has a fill mode
    ↪ , uses
                // the defaults from NetCDF
  esdm_container_t *c;
  // Some attributes provide information about the dataset as a whole
    ↪ and are
  // called global attributes. These are identified by the attribute
    ↪ name
  // together with a blank variable name (in CDL) or a special null "
    ↪ global
  // variable" ID (in C or Fortran).
  nc_dim_tbl_t dimt;
  md_vars_t vars;
  int parallel_mode;
#ifdef ESDM_PARALLEL
  MPI_Comm comm;
#endif
} nc_esdm_t;
```

```c
int ESDM_open(const char *path, int cmode, int basepe, size_t *
    ↪ chunksizehintp, void *parameters, struct NC_Dispatch *table, NC *ncp
    ↪ ) {
  const char *realpath = path;
  DEBUG_ENTER("%s\n", path);

  if (strncmp(path, "esdm:", 5) == 0) {
    realpath = &path[5];
  } else if (strncmp(path, "esd:", 4) == 0) {
    realpath = &path[4];
  }
  // remove leading slashes
  while (realpath[0] == '/') {
    realpath++;
  }
  char *cpath = strdup(realpath);
  // remove trailing slashes
  int pos = strlen(cpath) - 1;
  for (; pos > 0; pos--) {
    if (cpath[pos] != '/') {
      break;
    }
    cpath[pos] = '\0';
  }
  // const char * base = basename(realpath);

  DEBUG_ENTER("%s %d %d %s\n", cpath, ncp->ext_ncid, ncp->int_ncid, ncp->
    ↪ path);

  nc_esdm_t *e = malloc(sizeof(nc_esdm_t));
  memset(e, 0, sizeof(nc_esdm_t));
  e->ncid = ncp->ext_ncid;

  esdm_status status;

#ifdef ESDM_PARALLEL
  NC_MPI_INFO *data = (NC_MPI_INFO *)(parameters);
  if (data) {
    MPI_Comm_dup(data->comm, &e->comm);
    e->parallel_mode = 1;
    status = esdm_mpi_container_open(e->comm, cpath, 0, &e->c);
  } else {
    e->parallel_mode = 0;
    status = esdm_container_open(cpath, 0, &e->c);
  }
#else
  status = esdm_container_open(cpath, 0, &e->c);
#endif

  if (status != ESDM_SUCCESS) {
    return NC_EACCESS;
  }

  free(cpath);

  ncp->dispatchdata = e;

  // now build the dim table
  esdm_container_t *c = e->c;
  int ndsets = esdm_container_dataset_count(c);
  // open all ESDM datasets, find the names
  for (int i = 0; i < ndsets; i++) {
    esdm_dataset_t *dset = esdm_container_dataset_from_array(c, i);

#ifdef ESDM_PARALLEL
    if (e->parallel_mode) {
      status = esdm_mpi_dataset_ref(e->comm, dset);
    } else {
      status = esdm_dataset_ref(dset);
    }
#else
    status = esdm_dataset_ref(dset);
#endif
    if (status != ESDM_SUCCESS) {
      return NC_EINVAL;
    }

    esdm_dataspace_t *dspace;
    status = esdm_dataset_get_dataspace(dset, &dspace);
    if (status != ESDM_SUCCESS)
      return NC_EACCESS;
    int ndims = esdm_dataspace_get_dims(dspace);
    char const *const *names = NULL;
    status = esdm_dataset_get_name_dims(dset, &names);
    if (status != ESDM_SUCCESS) {
      return NC_EINVAL;
    }

    md_var_t *evar = malloc(sizeof(md_var_t));
    evar->dimidsp = malloc(sizeof(int) * ndims);
    evar->dset = dset;
    evar->fillmode = NC_FILL;

    int64_t const *dspace_size = esdm_dataset_get_size(dset);

    for (int j = 0; j < ndims; j++) {
      // check if the dim already exists in the dim table
      int dim_found = -1;
      for (int k = 0; k < e->dimt.count; k++) {
        if (strcmp(e->dimt.name[k], names[j]) == 0) {
          // found it!
          if (e->dimt.size[k] != dspace_size[j]) {
            WARN("Dimensions are not matching for %s", names[j]);
            return NC_EINVAL;
          }
          dim_found = k;
          break;
        }
      }
      if (dim_found == -1) {
        dim_found = e->dimt.count;
        add_to_dims_tbl(e, names[j], dspace_size[j]);
      }
      evar->dimidsp[j] = dim_found;
    }
    insert_md(&e->vars, evar);
  }

  smd_attr_t *attrs;
  status = esdm_container_get_attributes(e->c, &attrs);
  int dims_pos = smd_find_position_by_name(attrs, "_nc_dims");
  int sizes_pos = smd_find_position_by_name(attrs, "_nc_sizes");
  if (dims_pos >= 0 || sizes_pos >= 0) {
    if (dims_pos >= 0 && sizes_pos >= 0) {
      smd_attr_t *dims = smd_attr_get_child(attrs, dims_pos);
      smd_attr_t *sizes = smd_attr_get_child(attrs, sizes_pos);

      uint64_t count = smd_attr_elems(dims);
      if (smd_attr_elems(sizes) != count) {
        WARN("stored dimensions and sizes do not match, will ignore them");
      } else {
        char const **names = smd_attr_get_value(dims);
        uint64_t *dim_sizes = smd_attr_get_value(sizes);
        for (uint64_t i = 0; i < count; i++) {
          int dim_found = -1;
          for (int k = 0; k < e->dimt.count; k++) {
            if (strcmp(e->dimt.name[k], names[i]) == 0) {
              dim_found = k;
              break;
            }
          }
          if (dim_found == -1) {
            dim_found = e->dimt.count;
            // WARN("Adding unused dim: %s %lld", names[i], dim_sizes[i]);
            add_to_dims_tbl(e, names[i], dim_sizes[i]);
          }
        }
      }
    } else {
      WARN("stored only dimensions or sizes, will ignore them");
    }
    ncesdm_remove_attr(c);
  }

  return NC_NOERR;
}
```

Listing 2.3: Function ESDM_close

```c
int ESDM_close(int ncid, void *b) {
  DEBUG_ENTER("%d\n", ncid);
  nc_esdm_t *e = ESDM_nc_get_esdm_struct(ncid);
  if (e == NULL)
    return NC_EBADID;

  int ret = ncesdm_container_commit(e);

  int ndsets = esdm_container_dataset_count(e->c);
  // open all ESDM datasets, find the names
  for (int i = 0; i < ndsets; i++) {
    esdm_dataset_t *dset = esdm_container_dataset_from_array(e->c, i);
    esdm_dataset_close(dset);
  }
  ret = esdm_container_close(e->c);

  // TODO CLOSE the container properly
  free(e);
  return ret;
}
```

3. NetCDF4

NetCDF4 with Climate Forecast (CF) metadata has evolved to be the de-facto standard format for convenient data access for climate scientists.

It provides a set of features to enable convenient data access, for example, metadata can be used to assign names to variables, set units of measure, label dimensions, and provide other useful information. Portability allows data movement between different and possibly incompatible platforms, simplifying the exchange of data and facilitating communication between scientists. The ability to grow and shrink datasets, add new datasets and access small data ranges within datasets expedites the handling of data. Shared file access allows keeping the data in the same file, but unfortunately, it conflicts with performance and efficient usage of state-of-art HPC. Simultaneous access by several processes causes synchronization overhead which slows down I/O performance (synchronization is necessary to keep the data consistent).

The rapid development of computational power and storage capacity, and slow development of network bandwidth and I/O performance in the last years resulted in imbalanced HPC systems. Applications use the increased computational power to create more data. More data, in turn, requires more costly storage space, higher network bandwidth and sufficient I/O performance on storage nodes. But due to imbalance, the network and I/O performance are the main bottlenecks. This can be offset to some extent by using a part of the computational power for compression, adding a little extra latency for the transformation while significantly reducing the amount of data that needs to be transmitted or stored. However, before considering a compression method for HPC, it is a good idea to take a look at the realization of parallel I/O in modern scientific applications. Many of them use the NetCDF4 file format, which, in turn, uses HDF5 under the hood.

4. Typical NetCDF Data Mapping

Listing 4.1 gives an example for scientific metadata stored in a NetCDF file. Firstly, between Line 1 and 4, a few dimensions of the multidimensional data are defined. Here there are longitude, latitude with a fixed size and time with a variable size that allows to be extended (appending from a model). Then different variables are defined on one or multiple of the dimensions. The longitude variable provides a measure in “degrees east” and is indexed with the longitude dimension; in that case the variable longitude is a 1D array that contains values for an index between 0-479. It is allowed to define attributes on variables, this scientific metadata can define the semantics of the data and provide information about the data provenance. In our example, the unit for longitude is defined in Line 7. Multidimensional variables such as `sund` (Line 17) are defined on a 2D array of values for the longitude and latitude over various timesteps. The numeric values contain a scale factor and offset that has to be applied when accessing the data; since, here, the data is stored as short values, it should be converted to floating point data in the application. The `_FillValue` indicates a default value for missing data points.

Finally, global attributes such as indicated in Line 33 describe that this file is written with the NetCDF-CF schema and its history describes how the data has been derived / extracted from original data.

Listing 4.1: Example NetCDF metadata

```
dimensions:
  longitude = 480 ;
  latitude = 241 ;
  time = UNLIMITED ; // (1096 currently)
variables:
  float longitude(longitude) ;
    longitude:units = "degrees_east" ;
    longitude:long_name = "longitude" ;
  float latitude(latitude) ;
    latitude:units = "degrees_north" ;
    latitude:long_name = "latitude" ;
  int time(time) ;
    time:units = "hours since 1900-01-01 00:00:0.0" ;
    time:long_name = "time" ;
    time:calendar = "gregorian" ;

  short t2m(time, latitude, longitude) ;
    t2m:scale_factor = 0.00203513170666401 ;
    t2m:add_offset = 257.975148205631 ;
    t2m:_FillValue = -32767s ;
    t2m:missing_value = -32767s ;
    t2m:units = "K" ;
    t2m:long_name = "2 metre temperature" ;
  short sund(time, latitude, longitude) ;
    sund:scale_factor = 0.659209863732776 ;
    sund:add_offset = 21599.6703950681 ;
    sund:_FillValue = -32767s ;
    sund:missing_value = -32767s ;
    sund:units = "s" ;
    sund:long_name = "Sunshine duration" ;

// global attributes:
    :Conventions = "CF-1.0" ;
    :history = "2015-06-03 08:02:17 GMT by grib_to_netcdf-1.13.1:
   ↪ grib_to_netcdf /data/data04/scratch/netcdf-atls14-
   ↪ a562cefde8a29a7288fa0b8b7f9413f7-lFD4z9.target -o /data/data04/
   ↪ scratch/netcdf-atls14-a562cefde8a29a7288fa0b8b7f9413f7-CyGl1B.nc -
   ↪ utime" ;
}
```

5. NetCDF

NetCDF as important alternative is popular within the climate community. NetCDF provides a set of software libraries and self-describing, machine-independent, data formats that support the creation, access, and sharing of array-oriented scientific data. In the version 4 of the library (NetCDF4), the used binary file representation is HDF5. Like MPI and HDF5, NetCDF also defines its own set of atomic data types as shown in Table 5.1.

Similarly, to HDF5 and MPI, in addition to the atomic types the user can define his own types. NetCDF supports four different user defined types:

1. **Compound**: are a collection of types (either user defined or atomic)
2. **Variable Length Arrays**: are used to store non-uniform arrays
3. **Opaque**: only contain the size of each element and no data type information
4. **Enum**: like an enumeration in C

Once types are constructed, variables of the new type can be instantiated with `nc_def_var`. Data can be written to the new variable using `nc_put_var1`, `nc_put_var`, `nc_put_vara`, or `nc_put_vars`. Data can be read from the new variable with `nc_get_var1`, `nc_get_var`, `nc_get_vara`, or `nc_get_vars`. Finally, new attributes can be added to the variable using nc_put_att and existing attributes can be accessed from the variable using `nc_get_att`.

Table 5.2 shows the constructors provided to build user defined data types.

Type	Constructor	Description
	`nc_def_compound`	create a compound data type

Compound

Data type	Description
`NC_BYTE`	8-bit signed integer
`NC_UBYTE`	8-bit unsigned integer
`NC_CHAR`	8-bit character byte
`NC_SHORT`	16-bit signed integer
`NC_USHORT`	16-bit unsigned integer
`NC_INT`	32-bit signed integer
`NC_UINT`	32-bit unsigned integer
`NC_INT64`	64-bit signed integer
`NC_UINT64`	64-bit unsigned integer
`NC_FLOAT`	32-bit floating point
`NC_DOUBLE`	64-bit floating point
`NC_STRING`	variable length character string

Table 5.1.: netCDF Atomic External Data types

	`nc_insert_compound`	insert a name field into a compound data type
	`nc_insert_array_compound`	insert an array field into a compound data type
	`nc_inq_{compound,name,...}`	learn information about a compound data type
Variable Length Array	`nc_def_vlen`	create a variable length array
	`nc_inq_vlen`	learn about a variable length array
	`nc_free_vlen`	release memory from a variable length array
Opaque	`nc_def_opaque`	create an opaque data type
	`nc_inq_opaque`	learn about an opaque data type
Enum	`nc_def_enum`	create an enum data type
	`nc_insert_enum`	insert a named member into an enum data type
	`nc_inq_{enum,...}`	learn information about an enum data type

Table 5.2.: NetCDF Data types Constructors

6. Logical View: Component Overview

This section introduces a high-level overview for the earth system middleware (ESDM) and discusses the architecture according to the 4+1 model (refer to **??**). Chapter 6 starts with an architecture overview and introduces the core components and their relation to wide spread technologies related to I/O in earth system software. Chapter 6 covers the logical view, i.e, multiple aspects that affect how the ESDM operates and which semantics apply internally as well as externally to users/software using the ESDM. In particular, this includes the responsibilities of key components, the underlying data model and operations to manipulate the data including a mechanism to manage epochs. In **??**, the components are related to their physical location within the hardware and software stacks. **??**, discusses active components and processes required by the ESDM and how they interact when working concurrently.

The architecture overview provides only a brief description of the core components of the ESDM. We will refine the preliminary description of this document while we are building the prototype. While this document provides an overview of the ultimate system, within the ESiWACE project we will only be able to build a prototype for the central pieces. We seek to demonstrate the benefit of the middleware for the community to sustain development towards the presented vision. For more detailed descriptions in particular for the backends refer to **??**.

Problem Summary The ESD middleware has been designed to deal with the fact that existing data libraries for standardized data description and optimized I/O such as NetCDF, HDF5 and GRIB do not have suitable performance portable optimisations which reflect current data intensive system architectures and deliver cost-effective, acceptable data access bandwidth, latency and data durability.

The ESD Middleware To address these issues of performance portability, and exploiting exiting shared, interoperable interfaces, based on open standards, we have designed the *Earth System Data (ESD)* middleware (ESDM in short), which:

1. understands application data structures and scientific metadata, which lets us expose the same data via different APIs;
2. maps data structures to storage backends with different performance characteristics based on site specific configuration informed by a performance model;
3. yields best write performance via optimized data layout schemes that utilize elements from log-structured file systems;
4. provides relaxed access semantics, tailored to scientific data generation for independent writes, and;
5. includes a FUSE module which will provide backwards compatibility through existing

file formats with a configurable namespace based on scientific metadata.

Together these allow storing small and frequently accessed data on node-local storage, while serializing multi-dimensional data onto multiple storage backends – providing fault-tolerance and performance benefits for various access patterns at the same time. Compact-on-read instead of garbage collection will additionally optimize and replicate the data layout during reads via a background service. Additional tools allow data import/export for exchange between sites and external archives.

The ESDM aids the interests of stakeholders: developers have less burden to provide system specific optimizations and can access their data in various ways. Data centers can utilize storage of different characteristics.

A typical I/O-stack for an application using ESDM is shown in Figure 6.1. I/O of an existing application using the NetCDF (or HDF5) interface is processed by the ESDM plugin of HDF5 which may decide to store data on one of the available storage backends such as Lustre or an Object storage. Metadata may be stored in one of the supported metadata plugins. The user does not have to make decisions regarding the storage or metadata backends to be used; this decision is made by the middleware.

Details of the ESDM architecture is given in Figure 6.2. It provides more details about how ESDM is embedded into the existing software landscape and its high-level components:

Applications Use existing storage interfaces such as NetCDF4, GRIB or they may use the ESD interface.

Job scheduler The job scheduler assigns supercomputer resources to jobs. It may use the ESD interface to inform about future activity and stage/unstage data.

Middleware libraries are adjusted to be layered on top of the ESD interface.

ESD Interface This represents the API exposed to other libraries and users. The API is independent of the specific I/O backend used to store the data and supports structured

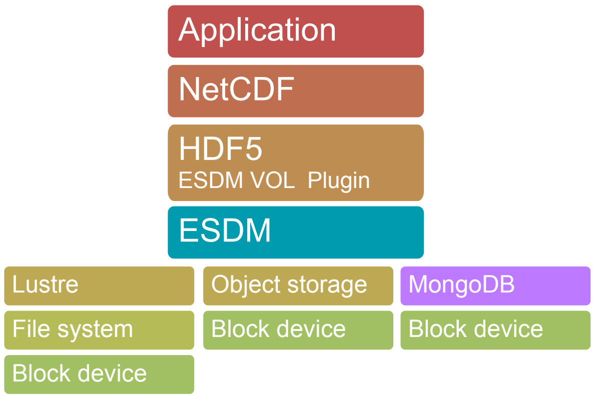

Figure 6.1.: A typical I/O-stack with the ESD middleware

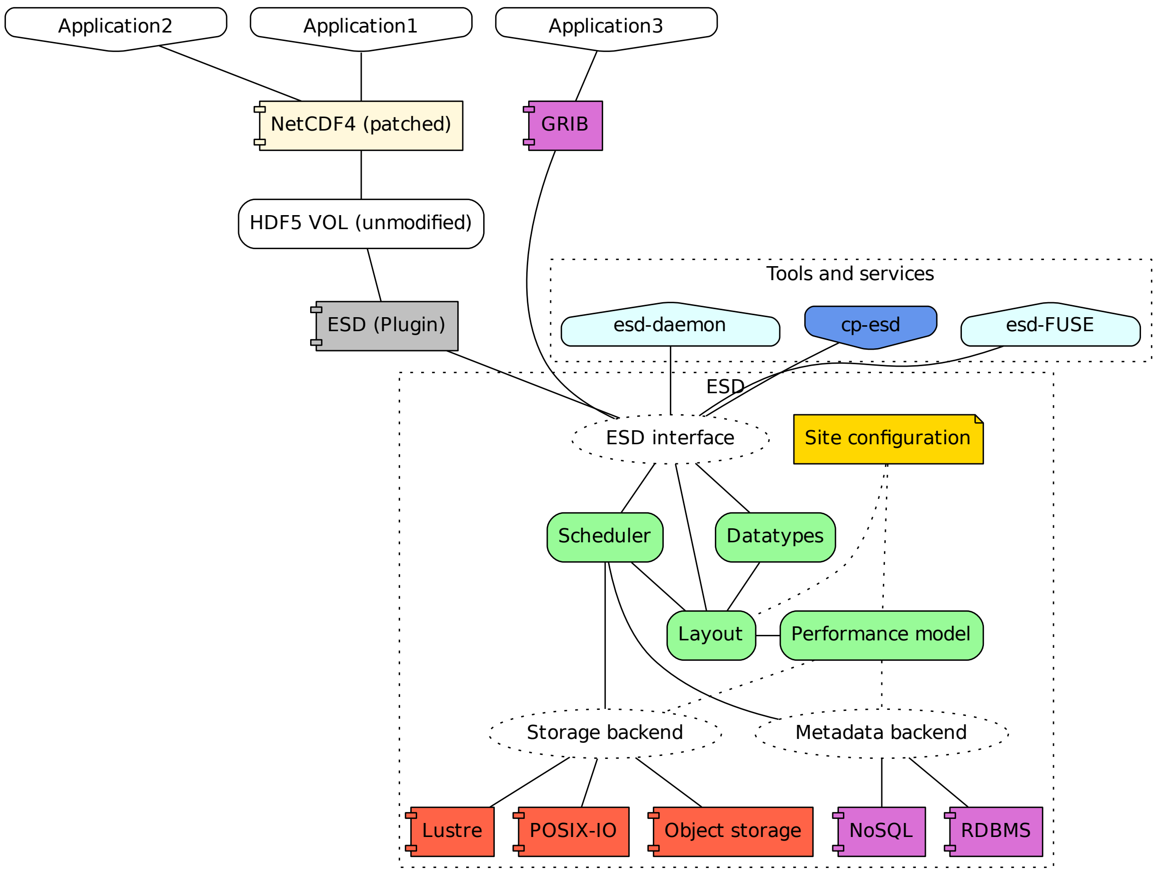

Figure 6.2.: Overview of the ESD architecture, relevant components and relationship to application and storage systems.

queries to perform complex data selections in the variables. The API is able to support the complex workflows of future applications.

Data types The data type component provides native data types that can be used by users or other libraries to describe data points inside variables. We follow the approach pursued by the MPI and HDF5 libraries, that is, we provide a set of native data types and a basic set of data type constructors that can be used to build custom derived data types.

Layout The layout component allows the middleware to store pieces of data on different backends depending on specific site configuration contained in the performance model. The layout component in this case takes responsibility for generating additional technical metadata describing data placement and for storing it in the appropriate metadata backend (i.e. MongoDB). A more detailed description of what technical metadata is, is given in the rest of this section.

Performance model This model predicts performance for data access using a site-specific configuration that describes the characteristics of available hardware technology. It is used by the layout component to make decisions of the data placement.

Scheduler The scheduler queues asynchronous calls from the API and processes them. It dispatches calls to storage and metadata backends and uses the layout component to identify beneficial placement of data.

Metadata backend Responsible to store all technical and scientific relevant metadata providing efficient access and manipulation.

Storage backend These backends are responsible to transform ESD objects and data structures to storage-technology-specific representations.

Tools and services On top of ESDM several user space tools are provided, a few examples are: The FUSE client provides backwards POSIX compatibility with existing applications. The daemon checks the consistency and integrity of the data managed by ESDM, potentially triggering actions to clean up and replicate data. The copy tool allows importing and exporting data from ESD to an existing storage infrastructure. It also serves as blue-print to embed its capabilities into higher-level tools such as GridFTP.

7. Logical View: Data Model

While data types introduced by computer architectures and software libraries are important for the data model, they are discussed separately in Section 7.4.

The data model of a system organizes elements of data, standardizes how they represent data entities and how users can interact with the data. The model can be split into three layers:

1. The **conceptual data model** describes the entities and the semantics of the domain that are represented by the data model and the typical operations to manipulate the data. In our case, the scientific domain is NWP/climate.

2. The **logical data model** describes the abstraction level provided by the system, how domain entities are mapped to objects provided by the system[1], and the supported operations to access and manipulate these objects are defined. Importantly, the **logical data model** defines the semantics when using the operations to access and manipulate the system objects. For example, a system object of a relational model is a table – representing attributes of a set of objects – and a row of a table representing attributes of a single object.

3. The physical data model describes how the logical entities are finally mapped to objects/files/regions on available hardware. The physical data model is partly covered by the backends of ESDM, therefore, the descriptions will stop at that abstraction level.

7.1. Conceptual Data Model

Our conceptual data model is aligned with the needs of domain scientists from climate and weather. It reuses and extends from concepts introduced in a data model proposed for the Climate and Forecasting conventions for NetCDF data[2].

Motivation from climate The ESDM needs to store, identify and manipulate data variables, V, containing scientific data from the (continuous) real or model world, discretised within a "sampling" domain d, that is

$$V = V(d)$$

where V may be air temperature, for instance. The domain d describes the location for each value of V, and is a function of its independent dimensions, for instance

$$d = d(Z(z), Y(y), X(x)))$$

[1] A entity of the domain model such as a car could be mapped to one or several objects.
[2] `"ACFDataModelandImplementation",Hasseletal,2017,GMDsubmitted`

where d is a three-dimensional domain described by axes of height, latitude, and longitude, sampled at coordinates found from the complete set of samples $Z(z), Y(y), X(x)$. Each set of coordinates x, y, z together specifies a location in the atmosphere at which V is specified.

The full dimensionality of the variable may exceed the number of dimensions needed to store it — for example, if V is air-temperature at 1.5m, then V may be sampled (and stored) in multiple 2-dimensional x, y arrays, with each additional array representing a different time step. In this case, some extra dimensions may be stored in metadata accompanying the scientific data.

Sampling may be regularly spaced along one or more of the dimensions, in which case the coordinates (e.g., x) of the samples can be found algorithmically from the dimensions (e.g., X), and we describe the coordinate-grid as "structured" in those dimensions; but they can also be irregularly spaced, and their individual positions may need to be stored, in which case we describe the grid as "unstructured" in those dimensions. With an unstructured grid it is not possible to fully understand the domain distribution of V unless all the coordinates are themselves stored as variables (e.g., $Z(z)$ is itself a variable defined at coordinates over a 1-dimensional domain spanning the height dimension). A coordinate grid may be structured in one set of dimensions and unstructured in another.

The values of V at the coordinate positions may represent a value at that point, or be representative of an area, volume or face of a cell defined in one or more of the dimensions.

In summary then, the conceptual (or scientific) data model consists of the following key entities:

Variable: A variable, V, defines a set of data representing a discrete (generally scalar) quantity discretised within a "sampling" domain, d. It is accompanied by

Metadata: which will include at the minimum, a name, but may also include units, and information about additional dimensionality, directly (e.g. via a key, value pair such as that necessary to expose $z = 1.5m$ for air temperature at 1.5m) or indirectly (e.g. via pointers to other generic coordinate variables which describe the sampled domain). There may also be a dictionary of additional metadata which may or may not conform to an external semantic convention or standard. Such metadata could include information about the tool used to observe or simulate the specific variable. Additional metadata is also required for all the other entities described below.

Dimension: The sampling domain d is defined by Dimensions which also defines coordinate axis. Dimensions will include metadata, which have to include at a minimum a name (e.g. height, time), but may also include information about directionality, units (e.g. degrees, months, days-since-a-particular-time-using-a-particular-calendar), or details of how to construct an algorithm to find the actual sampling coordinates, perhaps using a well-known algorithm such as an ISO 8601 time.

Coordinate: Coordinates are the set of values at which data is sampled along any given dimension. They may be explicitly defined by indexing into a coordinate variable, or implicitly defined by an algorithm. When we talk about the coordinates, it is usually clear if we mean the N-dimensional coordinate to address data in a given variable or if we just mean the (1D) coordinate along one dimension.

Cell: The data values are known at points, which may or may not represent a cell. Such cells are N-dimensional shapes where the dimensionality may or may not fully encompass the dimensionality of the domain. N-dimensional shapes can be implicitly defined in which case the Cartesian product of all dimensional coordinates forms the data "cube" of the cell, but they can also be explicitly defined, either by providing bounds on the coordinate variables (via metadata) or by introducing a new variable which explicitly defines the functional boundaries of the cell (as might happen in a finite element unstructured grid).

Dataset: Variables can be aggregated into datasets. A dataset contains multiple variables that logically belong together, and should be associated with metadata describing the reason for the aggregation. Variables must have unique names within a dataset.

Our conceptual model assumes that all variables are scalars, but clearly to make use of these scalars requires more complex interpretation.

Data type: which defines the types of values that are valid and the operations that can be conducted. While we are mostly dealing with scalars, they may not be amenable to interpretation as simple numbers. For example, a variable may be storing an integer which points into a taxonomy of categories of land-surface-types. More complex structures could include complex data types such as vectors, compressed ensemble values, or structures within this system, provided such interpretation is handled outside of the ESDM, and documented in metadata. This allows us to limit ourselves to simple data types plus arbitrary length blocks of bits.

Operators: Define the manipulations possible on the conceptual entities. The simplest operators will include creation, read, update and delete applied to an entity as a whole, or to a subset, however even these simple operators will come with constraints, for example, it should not be possible to delete a coordinate variable without deleting the parent variable as well. There will need to be a separation of concerns between operators which can be handled *within* the ESDM subsystem, and those which require external logic. Operators which might require external logic include subsetting — it will be seen that the ESDM will support simple subsetting using simple coordinates — but complex subsets such as finding a region in real space from dimensions spanned using an algorithm or coordinate variable, may require knowledge of how such algorithms or variables are specified. Such knowledge is embedded in conventions such as the CF NetCDF conventions, and this knowledge could only be provided to the ESDM via appropriate operator plugins.

Whatever the sampling regime and dimensionality, values of of a variable V will be laid out in storage. In the next section (7.2) we present the logical data model associated with the storage, before presenting a mapping of the conceptual data model to storage in section 7.3).

7.2. Logical Data Model

The logical data model describes how data is represented inside ESDM, the operations to interact with the data and their semantics. There are four first class entities in the ESDM logical data model: **variable**s, **fragment**s, **container**s, and **metadata**. ESDM entities may be linked by ESDM **reference**s, and a key property which emerges from the use of references is that no ESDM entity instance may be deleted while references to it still exist. The use of reference counting will ensure this property as well as avoid dangling pointers.

Figure 7.1 gives an overview of the logical data model.

Each of these entities is now described, along with a list of supported operations:

Variable: In the logical data model, the variable corresponds directly to a variable in the conceptual data model. Each element of the variable sampled across the dimensions contains data with a prescribed **DataType**. Variables are associated with both **Scientific Metadata** and **Technical Metadata**. Variables are partitioned into **fragments** each of which can be stored on one or more "storage backend". A variable definition includes internal information about the domain (bounding box in some coordinate system) and dimensionality (size and shape), while the detailed information about which coordinate variables are needed to span the dimensions and how they are defined is held in the technical metadata. Similarly, where a variable is itself a coordinate variable, a link to the parent variable for which it is used is held in the technical metadata. The ESDM will not allow an attempt to delete a variable to succeed while any such references exist (see references). A key part of the variable definition is the list of fragments associated with it, and if possible, how they are organised to span the domain. Users may choose to submit code pieces that are then run within the I/O-path (not part within ESiWACE implementation), such an operation covers the typical filter, map and reduce operations of the data flow programming paradigm.

Fragments are created by the backend while appending/modifying data to a variable.

Operations:

- Variables can be created and deleted.
- Fragments of data can be attached and deleted.
- Fragments can be repartitioned and reshuffled.
- Integrity can be checked.
- Data can be read, appended or modified those operations will be translated to the responsible fragments.
- Metadata can be atomically attached to a variable or modified.
- A variable can be sealed to make it immutable for all subsequent modifications.
- Process data of the variable somewhere in the I/O-path.

Fragment: A fragment is a piece (subdomain) of a variable. The ESDM expects to handle fragments as atomic entities, that is, only one process can write a fragment through the ESDM, and the ESDM will write fragments as atomic entities to storage backends. The backends are free to further partition these fragments as is appropriate, for example, by sharding using chunks as described in section 7.3. However, the ESDM is free to replicate fragments or subsets of fragments and to choose which backend is appropriate for any given fragment. This allows, for example, the ESDM to split a variable into fragments some of which are on stored on a parallel file system, while others are placed in object storage.

Operations:

- Data of fragments can be read, appended or modified.
- Integrity of the fragment can be checked.
- Process data of the variable somewhere in the I/O-path.

Container: A container is a virtual entity providing views on collections of variables, allowing multiple different datasets (as defined in the conceptual model) to be realised over the variables visible to the ESDM. Each container provides a hierarchical namespace holding references to one or multiple variables together with metadata.Variables cannot be deleted while they are referenced by a container. The use of these dynamic containers provides support for much more flexible organisation of data than provided by a regular file system semantics — and efficiently support high level applications such as the Data Reference Syntax[3].

A container provides the ESDM storage abstraction which is analogous to an external file. Because many scientific metadata conventions are based on semantic structures which span variables within a file in ways that may be opaque to the ESDM without the use of a plugin, the use of a container can indicate to the ESDM that these variables are linked even though the ESDM does not understand why, and so they cannot be independently deleted. When entire files in NetCDF format are ingested into the ESDM, the respective importing tool must create a container to ensure such linking properties are not lost.

Operations:

- Creation and deletion of containers.
- Creation and deletion of names in the hierarchical name space; the creation of links to an existing variable.
- Attaching and modification of metadata.
- Integrity can be checked.

Metadata: can be associated with all the other first class entities (variables, fragments, and containers). Such metadata is split into internal ESDM technical metadata, and external user-supplied semantic metadata. Technical metadata covers, for example, permissions, information about data location and timestamps. A backend will be able to add its own metadata to provide the information to lookup the data for the fragment from the storage

[3]Taylor et al (2012): CMIP5 Data Reference Syntax (DRS) and Controlled Vocabularies.

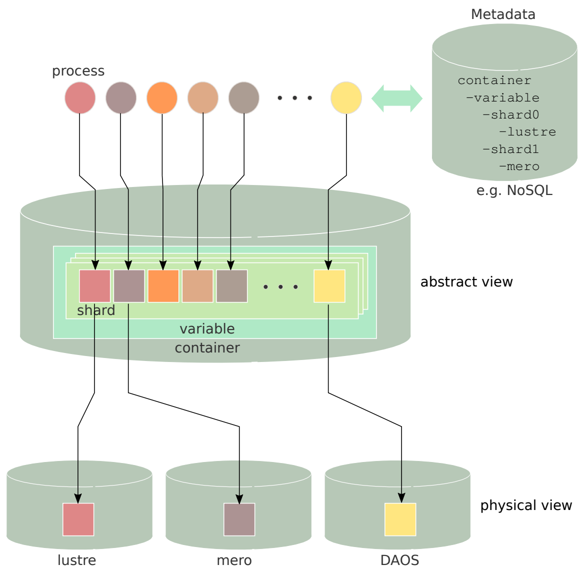

Figure 7.1.: Logical Data Model

backend managed by it. Metadata by itself is treated like a normal ESDM variable but linked to the variable of choice. The implementation may embed (simple) metadata into fragments of original data (see Reference).

Operations:

- Uses can create, read, or delete arbitrary scientific metadata onto variables and containers. A future version of the ESDM may support user scientific metadata for fragments.
- Container level metadata is generally not directly associated with variables, but may be retrieved via following references from variables to containers.
- Queries allow to search for arbitrary metadata, e.g., for objects that have (`experiment=X, model=Y, time=yesterday`) returning the variables and containers in a list that match. This enables to locate scientific data in an arbitrary namespace.

Reference A reference is a link between entities and can be used in many places, references can be embedded instead of real data of these logical objects. For example, dimensions inside a variable can be references, also a container typically uses references to variables.

Operations:

- A reference can be created from existing logical entities or removed.

Namespace ESDM does not offer a simple hierarchical namespace for the files. It provides the elementary functions to navigate data: teleportation and orientation in the following fashion: Queries about semantical data properties (e.g., `experiment=myExperiment, model=myModel, date=yesterday`) can be performed returning a list of matching files with their respective metadata. Iterating the list (orientation) is similar to listing a directory in a file system.

Note that this reduces the burden to define a hierarchical namespace and for data sharing services based on scientific metadata. An input/output container for an application can be assembled on the fly by using queries and name the resulting entities. As a container provides a hierarchical namespace, by harnessing this capability one can search for relevant variables and map them into the local file system tree, accessing these variables as if they would be, for example, NetCDF files. By offering a FUSE client, this feature also enables backwards compatibility for legacy POSIX applications.

7.3. Relationships between the Conceptual and Logical Data Model

The conceptual data model and logical data model are described above, and summarised in Figure 7.2. These UML and this version of the architecture do not fully deal with the issues around coordinate systems which are not described by simple monotonic coordinate arrays, for which simple bounding boxes can be constructed.

As noted above, to fully exploit more complicated coordinate systems it will be necessary to describe those coordinate systems more fully in the scientific metadata (LDM_Sci_Metadata) and potentially provide a plugin to the system to handle them. This notion is shown in the UML by virtue of the optional use of a named plugin to be identified in the scientific metadata, but the details of how that will work has been postponed to the prototype development.

The key high level entities are the conceptual container, variable, and domain, which have direct correspondence in the logical data model (see Figure 7.3) — however it is important to recognize that these are not isomorphic relationships. For example, the concept of a domain of a variable in the conceptual model is expanded in the logical data model to include subdomains associated with fragments, but the same class is used for both usages (LDM_Domain for both variable domain and fragment sub-domain).

7.4. Data types

The ESD middleware is specifically designed for weather and climate applications. These applications usually use GRIB and NetCDF as data format to send and store data. Nevertheless, the middleware should be also able to support other types of applications that use arbitrary libraries to represent and store data.

The NetCDF and HDF5 libraries define their own atomic and basic data types and then provide the APIs to build user defined data types from these. Generally speaking it makes sense to support a restricted number of most common data types that application can use out of the box and offer the possibility of extending these by means of additional APIs.

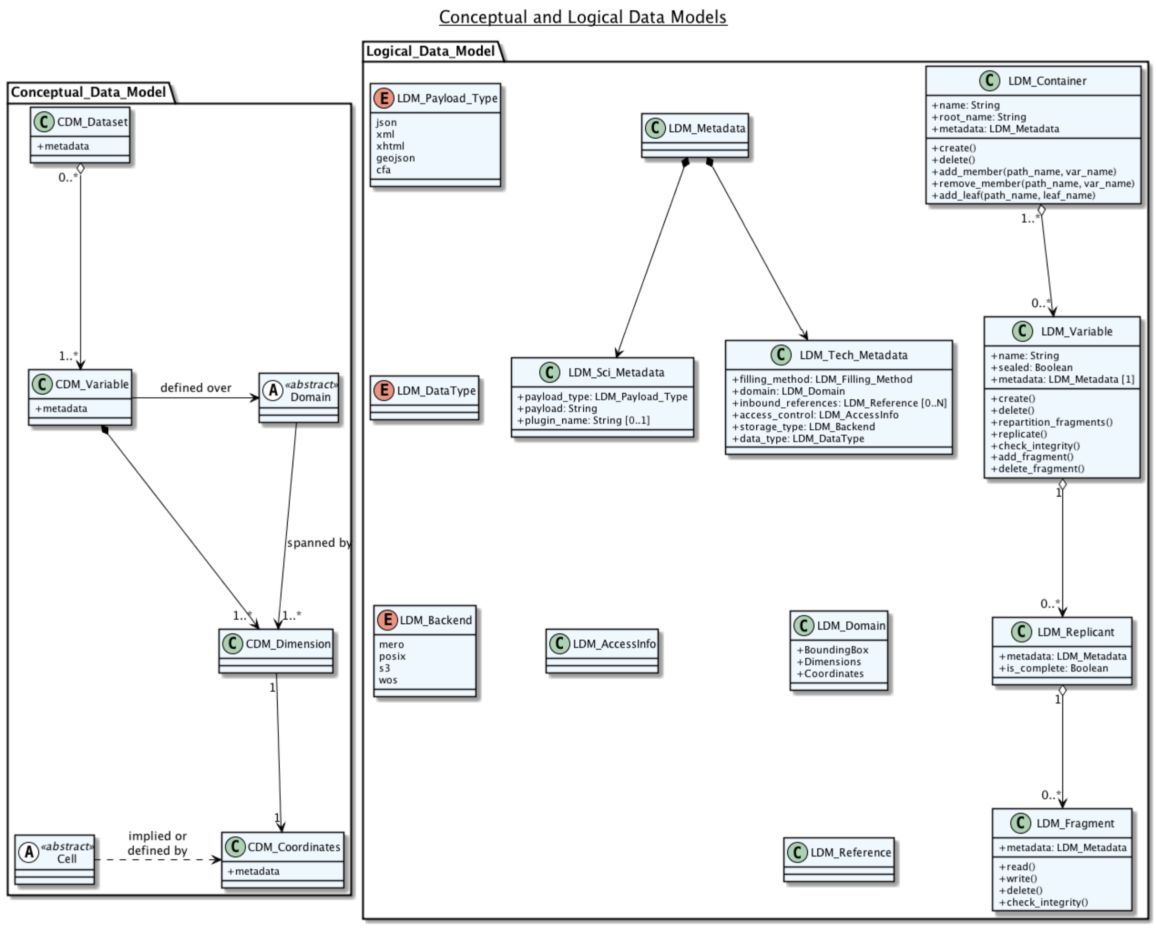

Figure 7.2.: A non-normative UML version of the conceptual and logical data models. The figure includes example for the operators of the invidiual logical operations. These UML are expected to be updated as the system is developed. See also figure 7.3.

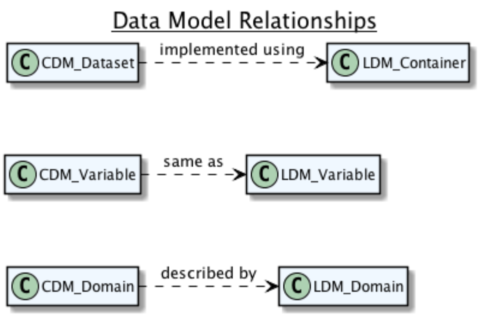

Figure 7.3.: Key relationships between conceptual and logical data models.

The support of native data types like H5T_NATIVE_INT or NC_INT is driven by the necessity to decouple the internal representation of the data from the way data is ultimately stored. Using native data types data can be correctly reconstructed passing from one representation to another. Like other libraries, the ESD middleware will also support a restricted range of native data types and a series of dedicated APIs to build user defined data types.

ESDM will support various atomic data types, integers, floating points, with different width, precision, endian and sign. The following table lists the possible atomic data types:

type	description
ESDM_T_I8	char
ESDM_T_U8	unsigned char
ESDM_T_I16	short (16bit)
ESDM_T_U16	unsigned short (16bit)
ESDM_T_I32	integer (32bit)
ESDM_T_U32	unsigned int (32bit)
ESDM_T_I64	long long (64bit)
ESDM_T_U64	unsigned long long (64bit)
ESDM_T_F16	float (16bit)
ESDM_T_F32	float (32bit)
ESDM_T_F64	double (64bit)
ESDM_T_F128	long double (128bit)
ESDM_T_TIMESTAMP	Date and time stamp

User defined complex data types include compound, variable length array, and array (fixed length array). Compound is similar to a struct in the C language. It is an aggregation of members, which are atomic data types or other complex data types. Array has fixed number of base data types, which are atomic data types or other complex data types. A variable-length array has a flexible length of base data types.

The definition of user defined data types have to be stored on back-end. When reading data from datasets, these data types definition is retrieved from back-end and parsed, and proper in-memory data structures and memories are allocated to accommodate the expected data points. Complex data types are encoded and stored on back-end in various formats according to different back-end. For example, for the Mero backend, these definition will be stored in Key/Value pair in index. The data type is integral part of the stored variable and fragment to allow the storage backend to understand the data and process it on demand (not part of the ESiWACE project). Figure 7.4 lists the required interfaces for compound, array and key value based data structures.

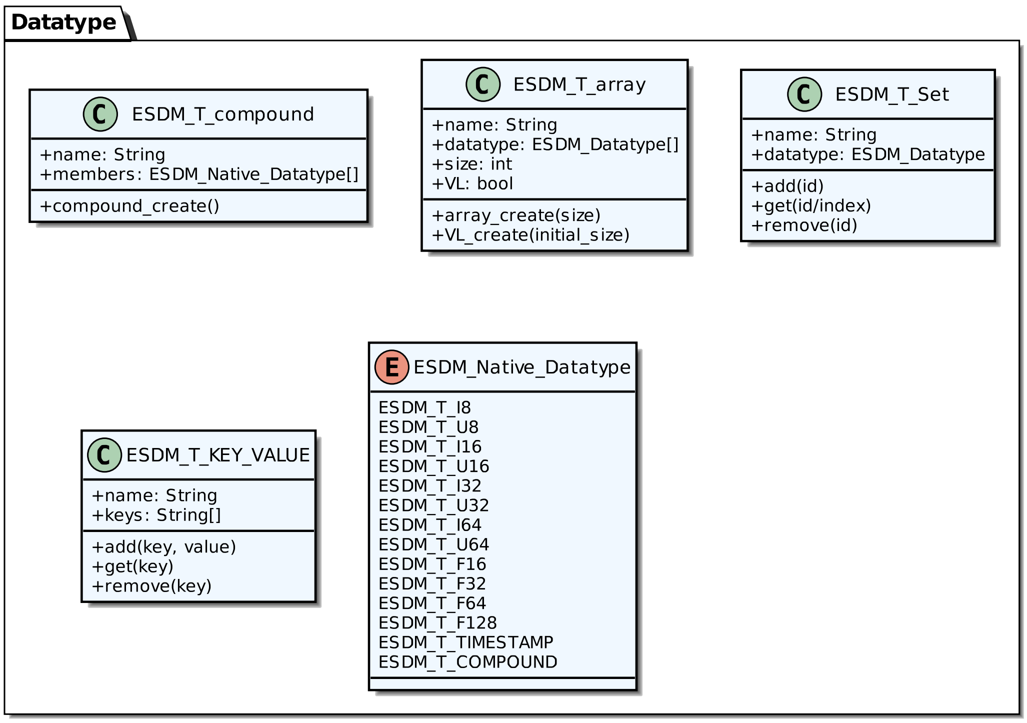

Figure 7.4.: Interfaces for the compound, array and key-value based data structures in relation to ESDM native data types.

8. NetCDF

This section describes the implications on an existing parallel application that uses one of the supported interfaces such as NetCDF4/HDF5. The semantics of the API calls will change slightly but typically in a way that is backwards compatible.

8.1. Logical View

This subsection covers dealing with filenames, opening and closing of containers as well as concurrency semantics. We assume an MPI parallelized application uses HDF5 with MPI support, e.g., parallel HDF5. The component diagram in Figure 8.1 illustrates how a HDF5/NetCDF ESDM frontend would mediate between the ESDM and an application. In addition, multiple processes using ESDM can coordinate using MPI, though only ESDM component using MPI is the ESDM HDF5 VOL Plugin.

8.1.1. Dealing with filenames

Traditionally, when opening a file with NetCDF the filename specifies the location, i.e., a URI where the data resides on a storage system. We change the notion of the filename to be the descriptor for a virtual container (virtual container descriptor). The virtual container can be composed of multiple URIs to integrate different variables into one virtual environment on the fly. Thus, from the reader's perspective it does not matter if data of a model is split into one or multiple physical files; upon read, all those files can be loaded together as if they would already exist in one logical file. It is also possible to avoid the use of the metadata backend; by specifying the locations of the variables on existing storage media, they can be

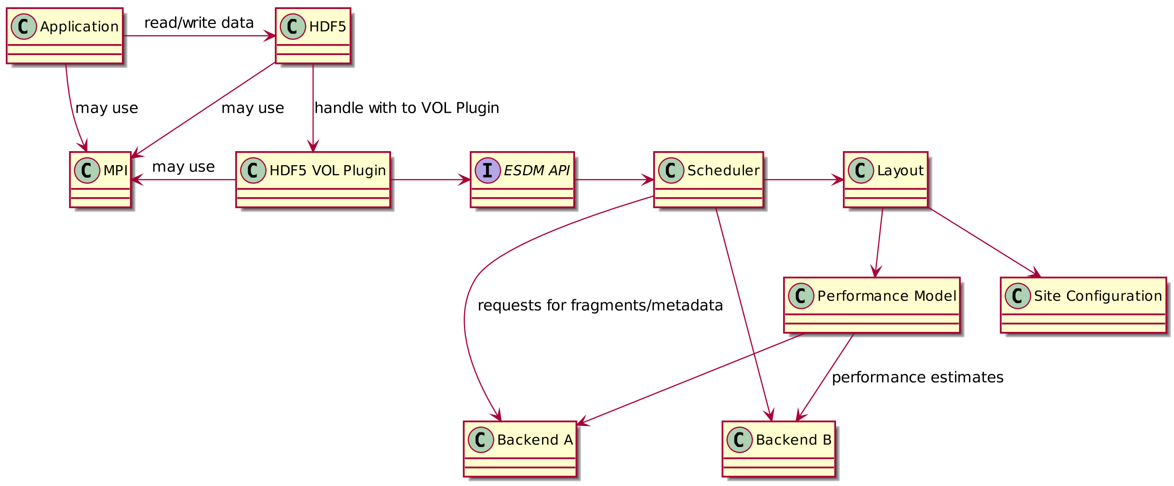

Figure 8.1.: Logical view to the HDF+MPI plugin.

linked into a virtual container. One restriction to this approach is the limitation of the length of filenames. To avoid this limitation, we support a prefix to the filename: `esd-cfg:/` that leads to a simple JSON file that contains the actual definition of the container.

8.1.2. Open

Opening a container (as defined by the filename) in ESDM will trigger the master process within the communicator to retrieve the necessary metadata from ESDM and broadcast it to all participating processes. Since the metadata is serializable to JSON, we can exchange the metadata easily.

8.1.3. Concurrency semantics

In general, the system is designed for parallel applications of which processes access data independently of each other. Still, metadata of internal objects such as containers and variables should be managed and updated explicitly by a single process of the application. That means, within one parallel (MPI) application, some kind of coordination must take place to allow the shared access to containers, variables and shards. A correct implementation for this behavior will be performed within the HDF5 VOL plugin.

Data sharing between independent applications is intended to happen after an epoch has been completed. It is not allowed that multiple parallel applications write data to the same variable at the same time. This is considered to be sufficient for most scenarios, e.g., a model produces some output; once the epoch completed, the produced data is post-processed.

8.1.4. Close

From the user perspective, closing a file that was opened in write mode, will make the content of the file visible in ESDM and durable for subsequent accesses. Thus, it updates the metadata, for example, incrementing the epoch of the variables and containers modified and updating the reference counters.

9. Metadata

9.1. Logical View

9.1.1. Metadata

There are two methods to include metadata, large metadata is includes as a reference to another variable containing the data, small metadata is embedded into the JSON of the MongoDB document.

Besides scientific metadata, the dynamic mapping of data to storage backends requires further metadata that must be managed. To distinguish technical metadata from scientific metadata, an internal namespace is created. Relevant technical metadata is shown in Table 9.1 for shards, variables and containers, respectively.

Metadata can be optional (O) or mandatory (M), and either is created automatically or must be set manually via the APIs. Automatic fields cannot be changed by the user. Some of the data can be automatically inferred, if not set manually, but manual setting may allow further optimizations.

Some of the metadata is used on several places, for example, information about the data lineage might be used to create several output variables. In our initial implementation, the metadata is stored redundantly because it 1) simplifies search; 2) enables us to restore data on corrupted storage systems by reading the metadata; 3) reduces contention and potentially false sharing of metadata. An implementation might decide to reduce this by utilizing normalized schemas.

References are the list of objects that are directly used by this object, e.g., other variables that are used to define the data further.

9.2. Mapping of metadata

To illustrate the applied mapping, we use a subset of our NetCDF metadata described in Chapter 4. The excerpt is given in Listing 9.1. The mapping of a single logical variable is exemplarily described in Listing 9.1.

Metadata	Field	Creation	Description
Domain	M	Auto	The subdomain this data covers from the variable
Type	M	Auto	The (potentially derived) data type of this shard
Variable	M	Auto	The ID of the varitechnical-on-metadata.texable this data belongs to
Storage	M	Auto	The storage backend used and its options
References	M	Auto	A list of objects that are referenced by this data
Sealed	M	Auto	A sealed shard is read-only

(a) For a shard

Metadata	Field	Creation	Description
Domain	M	Manual	Describes the overall domain
Type	M	Manual	The (potentially derived) data type
Info	M	Manual	The scientific metadata of this document
References	M	Auto	A list of objects that are referenced by this data
Permissions	M	Auto/Manual	The owner and permissions
Shards	M	Auto	The list of shard objects for this variable
Sealed	M	Auto	A sealed variable is read-only

(b) For a variable

Metadata	Field	Creation	Description
Owner	O	Manual	The owner of this file view (see the permission model)
Info	O	Manual	Additional scientific metadata for this view
Directory	O	Manual	Contains a mapping from names to variables
Environment	O	Automatic	Information about the application run
Permissions	M	Auto/Manual	The owner and permissions
References	M	Auto	A list of objects that are referenced by this data.

(c) For a container

Table 9.1.: Excerpt of additional technical metadata

Metadata	Description
Project	The scientific project during which the data is created
Institute	The institution which conducted the experiment
Person	A natural person; could be a contact, running the experiment
Contact	Reference to person or consortium
DOI	A document object identifier; useful for identifying a data publication
Topic	Some information about the topic of the data / experiment
Experiment	Description of this particular experiment
History	A list with the history and transformations conducted with the data

Table 9.2.: Excerpt of additional scientific metadata

Listing 9.1: NetCDF metadata for one variable

```
dimensions:
  longitude = 480 ;
  latitude = 241 ;
  time = UNLIMITED ; // (1096 currently)
variables:
  float longitude(longitude) ;
    longitude:units = "degrees_east" ;
    longitude:long_name = "longitude" ;
  float latitude(latitude) ;
    latitude:units = "degrees_north" ;
    latitude:long_name = "latitude" ;
  int time(time) ;
    time:units = "hours since 1900-01-01 00:00:0.0" ;
    time:long_name = "time" ;
    time:calendar = "gregorian" ;
  short sund(time, latitude, longitude) ;
    sund:scale_factor = 0.659209863732776 ;
    sund:add_offset = 21599.6703950681 ;
    sund:_FillValue = -32767s ;
    sund:missing_value = -32767s ;
    sund:units = "s" ;
    sund:long_name = "Sunshine duration" ;

// global attributes:
    :Conventions = "CF-1.0" ;
    :history = "2015-06-03 08:02:17 GMT by grib_to_netcdf-1.13.1:
    ↪ grib_to_netcdf /data/data04/scratch/netcdf-atls14-
    ↪ a562cefde8a29a7288fa0b8b7f9413f7-lFD4z9.target -o /data/data04/
    ↪ scratch/netcdf-atls14-a562cefde8a29a7288fa0b8b7f9413f7-CyGl1B.nc -
    ↪ utime" ;
}
```

To simplify search and identify data clearly, data services such as the WDCC and CERA, that offer data to the community, request scientists to provide additional metadata. Normally, such data is provided when the results of an experiment is ingested into such a database. Example metadata is listed in Table 9.2. In existing databases, the listed metadata is split into several fields, e.g. an address and email for persons, for simplicity only a rough overview is given. Instead of encoding the history as a simple text field, it could indicate detailed steps including the arguments for the commands and versions and transformations to reproduce the data. This should include for each step, where and the time when it is performed, and the versions of software used.

It is easily imaginable that most of this information could be useful already when the data is created as it simplifies the search and data management on the online storage. Some of the data fields become only available after the initial data creation, e.g., the DOI. Potentially the data must be updated / curated after the data is created.

9.3. Example

This example illustrates data of a predictive model could be stored on the system and the resulting metadata. The dimensionality of the underlying grid is fixed.

The application uses the following information to drive the simulation:

- Timerange: the simulated model time (from a starting datetime to the specified end)
- Longitude/Latitude: 1D data field with the coordinates [float]
- Temperature: Initial 2D field defined on (lon, lat)

A real model would use further parameters to estimate the temperature but these are sufficient to demonstrate the concepts. This information is either given as parameter to the simulation or read from an input (container). A mixture of both settings is possible.

The application produces the following output:

- Longitude/Latitude: 1D data field with the coordinates [float]
- Model time: the current datatime for the simulation
- Temperature: 2D field defined on (lon, lat, time) [float], containing the precise temperature on the coordinates defined by lon and lat for the given timestep
- AvgTemp: 1D field defined on (time) [float]; contains the mean temperature for the given time

9.3.1. Container

Upon application startup, we create a new virtual container that provide links to the already existing input. In Listing 9.2, the metadata for the container is shown, after the application is started. We assume it has used the APIs to provide the information (input, output, scientific metadata). In this example, we explicitly define the objects used as input; it is possible to also define the input as an already existing container. It is also possible to define the input a-priori if the objectIDs are known / looked up prior application run. The intended output variables could be given with their rough sizes. This would allow the scheduler to pre-stage the input and ensure that there is enough storage space available for the output. The environment information is inferred to the info object but can be changed from the user.

Listing 9.2: JSON document describing the container

```
"_id" : ObjectId(".."),
"directory" : {
  "input" : {
    "longitude" : ObjectId(".."),
    "latitude" : ObjectId(".."),
    "temperature" : ObjectId("..")
  },
  "output" {
    "temperature" : ObjectId(".."),
    "avgTemp" : ObjectId("..")
  }
},
"info" : { # This is the scientific metadata
  "model" : { "name" : "my model", "version" : "git ...4711" },
  "experiment" : {
    "tags"        : ["simulation", "poisson", "temperature"]
    "description" : "Trivial simulation of temperature using a poisson
↪  process"
  },
},
"environment" : {
  "date"  : datetime(2016, 12, 1),
  "system" : "mistral",
  "nodes" : ["m[1-1000]"]
},
"permissions" : {
  "UID"  : 1012,
  "GID"  : 400,
  "group" : "w", # allows read also
  "other" : "r"
},
"references" : {
  [ all links to used object IDs from input / output ]
}

```

9.3.2. Variable

The metadata for a single variable is build based on the information available in the container (such as permissions) and additional data provided by the user. Indeed, part of the metadata is replicated between container and variable as this preserves information about the creation of the variable that will typically not change during the lifetime of the variable.

An example for the temperature variable is shown in Listing 9.3. When describing the domain that is covered by the variable, there are three alternatives: 1) a reference to an existing variable is embedded and the minimum / maximum value is provided. This allows reusing descriptive information as data has to be stored only once. Min and max describe the multidimensional index of the subdomain in the variable that is actually referenced; 2) data becomes embedded into the file. This option is used when the size of the variable is small. An advantage of option 2) is that searches for data with a certain property do not require to lookup information in additional metadata.

Similarly, information about the data lineage (history) can originally be inferred from the objects linked in the directory mapping. In that case, the metadata of the referenced object must be copied, if the original object is removed.

3) A plugin for the interpretation and mapping of the coordinate system is used. The

plugin name must be stored and the respective parameters to identify the coordinates stored.

Listing 9.3: JSON document for temperature

```
"_id" : ObjectId("<TEMPID>"),
"sealed" : true,
"domain" : [
  "longitude" : [ "min" : 0, "max" : 359999, "reference" : ObjectId("..") ],
  "latitude" : [ "min" : 0, "max" : 179999, "reference" : ObjectId("..") ],
  "time" : [ datetime(...), datetime(...), ... ]
],
"type" : "float",
"info" : {
  "convention" : "CF-1.0",
  "name" : "temperature",
  "unit" : "K",
  "long description" : "This is the temperature",
  "experiment" : {
    "tags"         : ["simulation", "poisson", "temperature"]
    "description" : "Trivial simulation of temperature using a poisson process"
  },
  "model" : { "name" : "my model", "version" : "git ...4711" },
  "directory" : {
    "input" : {
      "longitude" : ObjectId("<LONID>"),
      "latitude" : ObjectId("<LATID>"),
      "temperature" : ObjectId("<TEMPID>")
    }
},
"environment" : {
  "date"     : datetime(2016, 12, 1),
  "system" : "mistral",
  "nodes"  : ["m[1-1000]"]
},
"history" : [
  ...
],

"permissions" : {
  "UID"  : 1012,
  "GID"  : 400,
  "group" : "w", # allows read also
  "other" : "r"
},
"references" : {
  [ all links to used object IDs ]
},
"shards" : [
  ObjectId(<SHARD1 ID>),
  # For a sealed object, the domains of its shards can optionally be embedded:
  { "reference" : ObjectId(<SHARD2 ID>), "storage" : ... , "domain" },
  ObjectId(<SHARD3 ID>),
  ObjectId(<SHARD4 ID>)
]
```

9.3.3. Shards

The variable is split into multiple shards; metadata for one of them is shown in Listing 9.4. Since we assume domain decomposition in the application, the longitude and latitude variables are now only partially stored in a shard. In the example, we assume two processes create one shard each and the surface of the earth is partitioned into four non-overlapping rectangles.

Listing 9.4: JSON document for a shard of the temperature variable

```
"_id" : ObjectId("<SHARD1 ID>"),
"sealed" : true,
"variable" : ObjectId("<TEMPID>"),
"type" : "float",
"domain" : {
  "longitude" : [ "min" : 0, "max" : 179999, "reference" : ObjectId(".."
    ↪ ) ],
  "latitude" : [ "min" : 0, "max" : 89999, "reference" : ObjectId("..")
    ↪ ],
  "time" : [ datetime(...), datetime(...), ... ]
},
"storage" : {
  "plugin" : "pfs",
  "options" : {
    "path" : "/mnt/lustre/testdir/file1",
  },
  "serialization" : "row-major"
},
"references : [
  ObjectId("<TEMPID>"),
  ObjectId(".."),
  ObjectId("..")
]
```

10. Introduction

This report is covering the compatibility between ESDM and NetCDF4. It is a supplementary document and not a formal deliverable of the ESiWACE project. We will update this document over the course of the development.

The Earth System Data Middleware (ESDM) provides a high level of abstraction for earth system applications in the presence of storage heterogeneity. This novel middleware aims to include aspects of workflow management capabilities to enable intelligent storage management that not only optimises data locality and performance but also lifts data management to a new level. The architecture utilises scientific metadata to exploit a centric perspective of the data structure while retaining well-established end-user interfaces. For further information on ESDM, check the Git Repository on `https://github.com/ESiWACE/esdm`.

The design goals of the ESDM are:

- Relaxed access semantics, tailored to scientific data generation
- Site-specific (optimised) data layout schemes
- Ease of use and deployment particularly configuration
- Enable a configurable namespace based on scientific metadata

NetCDF (Network Common Data Form) is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. The project homepage is hosted by the Unidata program at the University Corporation for Atmospheric Research (UCAR). The format is an open standard. The project started in 1989 and is still actively supported by UCAR. The original NetCDF binary format (released in 1997, now known as **NetCDF classic format**) is still widely used across the world and continues to be fully supported in all NetCDF releases.

NetCDF is also a community standard for sharing scientific data. The Unidata Program Center supports and maintains NetCDF programming interfaces for C, C++, Java, and Fortran. Programming interfaces are also available for Python, IDL, MATLAB, R, Ruby, and Perl. Version 4.0 (released in 2008) allowed the use of the HDF5 data file format. Version 4.1 (2010) added support for C and Fortran client access to specified subsets of remote data via OPeNDAP. Version 4.3.0 (2012) added a CMake build system for Windows builds. Version 4.7.0 (2019) added support for reading Amazon S3 objects. Further releases are planned to improve performance, add features, and fix bugs. For further information on NetCDF, check `https://www.unidata.ucar.edu/software/netcdf/`.

Data in NetCDF format is:

Self-Describing A NetCDF file includes information about the data it contains.

Portable A NetCDF file can be accessed by computers with different ways of storing integers, characters, and floating-point numbers.

Scalable Small subsets of large datasets in various formats may be accessed efficiently through NetCDF interfaces, even from remote servers.

Appendable Data may be appended to a properly structured NetCDF file without copying the dataset or redefining its structure.

Sharable One writer and multiple readers may simultaneously access the same NetCDF file.

Archivable Access to all earlier forms of NetCDF data will be supported by current and future versions of the software.

10.1. Outline

This report is organised as follows:

Chapter 11 describes the main NetCDF funcionalities and their coverage by ESDM.

Chapter 12 introduces the C tests provided by NetCDF in its documentation and the status when running with ESDM.

Chapter 13 introduces the Python tests provided by NetCDF in its documentation and the status when running with ESDM.

Chapter 14 analyses the NetCDF Performance Benchmark Tool.

Chapter 15 explores the nccopy utility.

Chapter 17 concludes this compatibility report.

11. NetCDF and ESDM Functionalities

This chapter compares the NetCDF functionalities with the current version of ESDM.

11.1. Error Handling

Each NetCDF function returns 0 on success. In general, if a function returns an error code, you can assume it did not do what you hoped it would. NetCDF functions return a non-zero status codes on error. When programming with NetCDF, the return code of every NetCDF API call can be looked up in `netcdf.h` (for C programmers).

Each NetCDF function returns an integer status value. If the returned status value indicates an error, you may handle it in any way desired, from printing an associated error message and exiting to ignoring the error indication and proceeding (this is not recommended!).[1]

11.1.1. ESDM

NetCDF has an extense classification for the possible errors that might happen. ESDM does not share this classification, and it is something that their developers are not considering to include in the final version. This decision does not affect the performance of ESDM, but it was critical when the NetCDF tests were evaluated. NetCDF tests introduce several wrong conditions, and ESDM does not produce the expected error. Because of that, the code in the NetCDF tests that considers invalid parameters as input was removed.

11.2. Classic Model

The classic NetCDF data model consists of variables, dimensions, and attributes. This way of thinking about data was introduced with the very first NetCDF release, and is still the core of all NetCDF files.

Variables N-dimensional arrays of data. Variables in NetCDF files can be one of six types (char, byte, short, int, float, double).

Dimensions describe the axes of the data arrays. A dimension has a name and a length. An unlimited dimension has a length that can be expanded at any time, as more data are written to it. NetCDF files can contain at most one unlimited dimension.

[1]Reference: `https://www.unidata.ucar.edu/software/netcdf/docs/group__error.html`

Attributes annotate variables or files with small notes or supplementary metadata. Attributes are always scalar values or 1D arrays, which can be associated with either a variable or the file as a whole. Although there is no enforced limit, the user is expected to keep attributes small.

With NetCDF-4, the NetCDF data model has been extended, in a backwards compatible way. The new data model, which is known as the **Common Data Model** is part of an effort here at Unidata to find a common engineering language for the development of scientific data solutions. It contains the variables, dimensions, and attributes of the classic data model, but adds:

Groups A way of hierarchically organizing data, similar to directories in a Unix file system.

User-defined Types The user can now define compound types (like C structures), enumeration types, variable length arrays, and opaque types.

These features may only be used when working with a NetCDF-4/HDF5 file. Files created in classic formats cannot support groups or user-defined types.[2]

11.2.1. ESDM

NetCDF includes tests with the Classic Model; while the ESDM data model basically supports all features of the Classic Model, it does not support the setting of the mode.

11.3. Modes

Some tests consider in which mode the file is open. There are two modes associated with accessing a NetCDF file:[3]

Define Mode In define mode, dimensions, variables, and new attributes can be created, but variable data cannot be read or written.

Data Mode In data mode, data can be read or written, and attributes can be changed, but new dimensions, variables, and attributes cannot be created.

11.3.1. ESDM

The current version of ESDM does not have restrictions regarding the modes. Once the file is open, the user can do any modifications s/he wants. Tables 11.1 amd 11.3 compares the options for creating and opening a file using NetCDF and ESDM.

[2]Reference: `https://www.unidata.ucar.edu/software/netcdf/docs/netcdf_data_model.html`
[3]Reference: `https://northstar-www.dartmouth.edu/doc/idl/html_6.2/NetCDF_Data_Modes.html`

FLAG	NetCDF Support	ESDM Support
FLAG	NetCDF Support	ESDM Support[4]
NC_CLOBBER	Overwrite existing file	ESDM_CLOBBER
NC_NOCLOBBER	Do not overwrite existing file	ESDM_NOCLOBBER
NC_SHARE	Limit write caching - netcdf classic files only	NOT SUPPORTED
NC_64BIT_OFFSET	Create 64-bit offset file	NOT SUPPORTED
NC_64BIT_DATA	Create CDF-5 file (alias NC_CDF5)	NOT SUPPORTED
NC_NETCDF4	Create NetCDF-4/HDF5 file	NOT SUPPORTED
NC_CLASSIC_MODEL	Enforce NetCDF classic mode on NetCDF-4/HDF5 files	NOT SUPPORTED
NC_DISKLESS	Store data in memory	NOT SUPPORTED
NC_PERSIST	Force the NC_DISKLESS data from memory to a file	NOT SUPPORTED

Table 11.1.: Modes – Creating a file.

FLAG	NetCDF Support	ESDM Support
NC_NOWRITE	Open the dataset with read-only access	ESDM_MODE_FLAG_READ
NC_WRITE	Open the dataset with read-write access	ESDM_MODE_FLAG_WRITE
NC_SHARE	Share updates, limit caching	NOT SUPPORTED
NC_DISKLESS	Store data in memory	NOT SUPPORTED
NC_PERSIST	Force the NC_DISKLESS data from memory to a file	NOT SUPPORTED

Table 11.2.: Modes – Opening a file.

11.4. Data Types

Data in a NetCDF file may be one of the **external data types** (Section 11.4.1), or may be a **user-defined data type** (Section 11.4.2).[5]

11.4.1. External Data Types

The atomic external types supported by the NetCDF interface are:

[5]Reference: `https://www.unidata.ucar.edu/software/netcdf/docs/data_type.html`

Type	Description
NC_BYTE	8-bit signed integer
NC_UBYTE	8-bit unsigned integer
NC_CHAR	8-bit character
NC_SHORT	16-bit signed integer
NC_USHORT	16-bit unsigned integer
NC_INT (or NC_LONG)	32-bit signed integer
NC_UINT	32-bit unsigned integer
NC_INT64	64-bit signed integer
NC_UINT64	64-bit unsigned integer
NC_FLOAT	32-bit floating-point
NC_DOUBLE	64-bit floating-point
NC_STRING	variable length character string

Table 11.3.: Atomic external types.

11.4.2. User-Defined Types

The user can now define compound types (like C structures), enumeration types, variable length arrays, and opaque types[6].

Compound Types In NetCDF-4 files it is possible to create a data type which corresponds to a C struct. These are known as **compound** types (following HDF5 nomenclature).

That is, a NetCDF compound type is a data structure which contains an arbitrary collection of other data types, including other compound types.

To define a new compound type, use `nc_def_compound()`. Then call `nc_insert_compound()` for each type within the compound type.

Read and write arrays of compound data with the `nc_get_vara()` and `nc_put_vara()` functions. These functions were actually part of the NetCDF-2 API, brought out of semi-retirement to handle user-defined types in NetCDF-4.

Opaque Types Store blobs of bits in opaque types. Create an opaque type with nc_def_opaque. Read and write them with `nc_get_vara()`/`nc_put_vara()`.

Variable Length Arrays (VLEN) Create a VLEN type to store variable length arrays of a known base type. Use `nc_def_vlen()` to define a VLEN type, read and write them with `nc_get_vara()`/`nc_put_vara()`.

[6] References: https://www.unidata.ucar.edu/software/netcdf/docs/user_defined_types.html and https://www.unidata.ucar.edu/software/netcdf/netcdf/NetCDF-user-defined-data-types.html

11.4.3. ESDM

Number	NetCDF TYPE	ESDM Type	ESDM Representation
1	NC_BYTE	SMD_DTYPE_INT8	int8_t
2	NC_CHAR	SMD_DTYPE_CHAR	char
3	NC_SHORT	SMD_DTYPE_INT16	int16_t
4	NC_INT	SMD_DTYPE_INT32	int32_t
4	NC_LONG	SMD_DTYPE_INT32	int32_t
5	NC_UINT64	SMD_DTYPE_UINT64	uint64_t
6	NC_DOUBLE	SMD_DTYPE_DOUBLE	64 bits
7	NC_UBYTE	SMD_DTYPE_UINT8	uint8_t
8	NC_USHORT	SMD_DTYPE_UINT16	uint16_t
9	NC_UINT	SMD_DTYPE_UINT32	uint32_t
10	NC_INT64	SMD_DTYPE_INT64	int64_t
11	NC_FLOAT	SMD_DTYPE_FLOAT	32 bits

Table 11.4.: Equivalence between ESDM and NetCDF4 data types.

The current version of ESDM does not support user-defined data types, but the developers intend to support this feature in the final version. Table 11.5 summarizes the available NetCDF data types and the corresponding support from ESDM.

FLAG	NetCDF Support	ESDM Support
NC_NAT	NAT = Not A Type (c.f. NaN)	SMD_TYPE_AS_EXPECTED
NC_BYTE	signed 1 byte integer	SMD_DTYPE_INT8
NC_CHAR	ISO/ASCII character	SMD_DTYPE_CHAR
NC_SHORT	signed 2 byte integer	SMD_DTYPE_INT16
NC_INT	signed 4 byte integer	SMD_DTYPE_INT32
NC_LONG	deprecated, but required for backward compatibility	SMD_DTYPE_INT32
NC_FLOAT	single precision floating-point number	SMD_DTYPE_FLOAT
NC_DOUBLE	double precision floating-point number	SMD_DTYPE_DOUBLE
NC_UBYTE	unsigned 1 byte int	SMD_DTYPE_UINT8
NC_USHORT	unsigned 2-byte int	SMD_DTYPE_UINT16
NC_UINT	unsigned 4-byte int	SMD_DTYPE_UINT32
NC_INT64	signed 8-byte int	SMD_DTYPE_INT64
NC_UINT64	unsigned 8-byte int	SMD_DTYPE_UINT64
NC_STRING	string	SMD_DTYPE_STRING
NC_VLEN	used internally for vlen types	NOT SUPPORTED YET
NC_OPAQUE	used internally for opaque types	NOT SUPPORTED YET
NC_COMPOUND	used internally for compound types	NOT SUPPORTED YET
NC_ENUM	used internally for enum types	NOT SUPPORTED YET

Table 11.5.: Data Types Support.

11.5. Compression

The NetCDF-4 libraries inherit the capability for data compression from the HDF5 storage layer underneath the NetCDF-4 interface. Linking a program that uses NetCDF to a NetCDF-4 library allows the program to read compressed data without changing a single line of the program source code. Writing NetCDF compressed data only requires a few extra statements. And the nccopy utility program supports converting classic NetCDF format data to or from compressed data without any programming[7].

11.5.1. ESDM

ESDM does not support compression yet. Because of that, all functions and tests related to chunking, deflate, and fletcher will not work when using ESDM. We will integrate a compression library in the future and support quantification of error tolerance levels for different variables, but it was not yet integrated with the current version of ESDM. The Scientific Compression Library (SCIL) can be found in the following Git Repository:

`https://github.com/JulianKunkel/scil/`

11.6. Endianness

The endianness is defined as the order of bytes in multi-byte numbers: numbers encoded in big endian have their most significant bytes written first, whereas numbers encoded in little endian have their least significant bytes first. Little-endian is the native endianness of the IA32 architecture and its derivatives, while big-endian is native to SPARC and PowerPC, among others[8].

ESDM only supports native-endianness of the machine it runs on. The rationale behind this design choice is that ESDM will be deployed in data centers and will be used to store data optimally in the data center partitioned across available storage solutions. It is not intended to be stored in a portable fashion, therefore, data can be imported/exported between, e.g., a NetCDF format and ESDM native format.

11.6.1. ESDM

ESDM only supports native-endianness of the machine it runs on. The developers believe that the native-endianness of the machine is enough for demonstrating the benefits of using ESDM to improve efficiency in the system.

[7]Reference: `https://www.unidata.ucar.edu/blogs/developer/entry/netcdf_compression`
[8]Reference: `https://www.gnu.org/software/guile/manual/html_node/Bytevector-Endianness.html`

11.7. Groups

NetCDF-4 files can store attributes, variables, and dimensions in hierarchical groups. This allows the user to create a structure much like a Unix file system. In NetCDF, each group gets an ncid. Opening or creating a file returns the ncid for the root group (which is named /). Dimensions are scoped such that they are visible to all child groups. For example, you can define a dimension in the root group, and use its dimension id when defining a variable in a sub-group. Attributes defined as NC_GLOBAL apply to the group, not the entire file. The degenerate case, in which only the root group is used, corresponds exactly with the classic data mode, before groups were introduced[9].

11.7.1. ESDM

In general, ESDM does not support groups from NetCDF. When only the root group is used, ESDM can work adequately and assumes the group and the file are the same entity.

The ability to work with groups is a functionality that ESDM developers may implement depending on future requirements.

11.8. Fill Values

Sometimes there are missing values in the data, and some value is needed to represent them. For example, what value do you put in a sea-surface temperature variable for points over land? In NetCDF, you can create an attribute for the variable (and of the same type as the variable) called _FillValue that contains a value that you have used for missing data. Applications that read the data file can use this to know how to represent these values[10].

11.8.1. ESDM

ESDM supports fill values. There are some speficic details in the implementation of fill values inside ESDM that worth noticing.

TO CHECK

11.9. Type Conversion

Each NetCDF variable has an external type, specified when the variable is first defined. This external type determines whether the data is intended for text or numeric values, and if numeric, the range and precision of numeric values.

If the NetCDF external type for a variable is char, only character data representing text strings can be written to or read from the variable. No automatic conversion of text data to

[9]Reference: `https://www.unidata.ucar.edu/software/netcdf/docs/groups.html`
[10]Reference: `https://www.unidata.ucar.edu/software/netcdf/docs/fill_values.html`

a different representation is supported. If the type is numeric, however, the NetCDF library allows you to access the variable data as a different type and provides automatic conversion between the numeric data in memory and the data in the NetCDF variable. For example, if you write a program that deals with all numeric data as double-precision floating-point values, you can read NetCDF data into double-precision arrays without knowing or caring what the external type of the NetCDF variables are. On reading NetCDF data, integers of various sizes and single-precision floating-point values will all be converted to double-precision, if you use the data access interface for double-precision values. Of course, you can avoid automatic numeric conversion by using the NetCDF interface for a value type that corresponds to the external data type of each NetCDF variable, where such value types exist.

The automatic numeric conversions performed by NetCDF are easy to understand, because they behave just like assignment of data of one type to a variable of a different type. For example, if you read floating-point NetCDF data as integers, the result is truncated towards zero, just as it would be if you assigned a floating-point value to an integer variable. Such truncation is an example of the loss of precision that can occur in numeric conversions.

Note that mere loss of precision in type conversion does not result in an error. For example, if you read double precision values into an integer, no error results unless the magnitude of the double precision value exceeds the representable range of integers on your platform. Similarly, if you read a large integer into a float incapable of representing all the bits of the integer in its mantissa, this loss of precision will not result in an error. If you want to avoid such precision loss, check the external types of the variables you access to make sure you use an internal type that has a compatible precision.

Converting from one numeric type to another may result in an error if the target type is not capable of representing the converted value. For example, an integer may not be able to hold data stored externally as an IEEE floating-point number. When accessing an array of values, a range error is returned if one or more values are out of the range of representable values, but other values are converted properly[11].

The NC_ERANGE error is returned by any of the reading or writing functions when one or more of the values read or written exceeded the range for the type. (For example if you were to try to read 1000 into an unsigned byte.) In the case of NC_ERANGE errors, the NetCDF library completes the read/write operation, and then returns the error. The type conversion is handled like a C type conversion, whether or not it is within range. This may yield bad data, but the NetCDF library just returns NC_ERANGE and leaves it up to the user to handle. (For more information about type conversion see Type Conversion)[12].

TO CHECK

Whether a range error occurs in writing a large floating-point value near the boundary of representable values may be depend on the platform. The largest floating-point value you can write to a NetCDF float variable is the largest floating-point number representable on your system that is less than 2 to the 128th power. The largest double precision value you can write to a double variable is the largest double-precision number representable on your system that is less than 2 to the 1024th power.

[11]Reference: `https://www.unidata.ucar.edu/software/netcdf/docs/data_type.html#type_conversion`
[12]Reference: `https://www.unidata.ucar.edu/software/netcdf/docs/group__error.html`

11.9.1. ESDM

ESDM supports most of the data conversions but may return a slightly different error.

ESDM deals with type conversion the same way as NetCDF. However, ESDM only accept conversions for attributes, and not for variables. The reason behind this choice is ...

TO CHECK

12. The NetCDF Tests

The directory **nc_test4** has originally 104 tests. The tests were classified according to their output using NetCDF. Basically, we expect that all benchmarks run with NetCDF successfully but some tests turned out to be non-functional. There are four categories for the tests:

Not Tested It means the test was not tested. This category usually represents tests that are too long to run or tests that demand a large amount of memory or processing.

Not Building It means it was not possible to compile the test.

Not Running It means the test compiles, but there is something wrong when the test runs and the expected output is not produced.

Working It means the test compiles, runs and produces the expected output.

Tables 12.1, 12.2 and 12.3 presents the classification for the tests.

File	Not Tested	Not Building	Not Running	Working
bigmeta.c	✓			
bm_file.c		✓		
bm_many_atts.c		✓		
bm_many_objs.c		✓		
bm_netcdf4_recs.c		✓		
cdm_sea_soundings.c				
h5testszip.c				
openbigmeta.c	✓			
ref_bzip2.c		✓		
renamegroup.c				✓
test_filter.c		✓		
test_filter_misc.c		✓		
test_szip.c				✓
tst_ar4_3d.c		✓		
tst_ar4_4d.c		✓		
tst_ar4.c		✓		
tst_atts1.c				✓
tst_atts2.c				✓
tst_atts3.c				✓
tst_atts.c				✓
tst_atts_mod.c				
tst_attsperf.c				✓
tst_atts-simple.c				
tst_atts_string_rewrite.c				✓
tst_bug324.c				✓

Table 12.1.: List of nc_test4 files – Part I

File	Not Tested	Not Building	Not Running	Working
tst_camrun.c				✓
tst_chunks2.c				✓
tst_chunks3.c				✓
tst_chunks.c				✓
tst_compounds2.c				✓
tst_compounds3.c				✓
tst_compounds.c			✓	
tst_converts2.c				✓
tst_converts.c				✓
tst_coords2.c				✓
tst_coords3.c				✓
tst_coords.c				✓
tst_create_files.c				✓
tst_dims2.c				✓
tst_dims3.c				✓
tst_dims.c			✓	
tst_elatefill.c				✓
tst_empty_vlen_unlim.c				✓
tst_endian_fill.c				✓
tst_enums.c				✓
tst_files2.c		✓		
tst_files3.c				✓
tst_files4.c				✓
tst_files5.c				✓
tst_files6.c				✓
tst_files.c				✓
tst_fill_attr_vanish.c				✓
tst_fillbug.c				✓
tst_fills2.c				✓
tst_fills.c				✓
tst_filterparser.c				✓
tst_grps2.c				✓
tst_grps.c				✓
tst_h5_endians.c				✓
tst_hdf5_file_compat.c				✓
tst_h_many_atts.c		✓		
tst_h_refs.c				✓
tst_h_scalar.c				✓
tst_h_strbug.c				✓
tst_interops4.c			✓	
tst_interops5.c				✓
tst_interops6.c				✓
tst_interops.c				✓
tst_knmi.c		✓		
tst_large2.c				✓
tst_large.c				✓
tst_mem.c				✓

Table 12.2.: List of nc_test4 files – Part II

File	Not Tested	Not Building	Not Running	Working
tst_misc.c				
tst_mode.c				✓
tst_mpi_parallel.c				✓
tst_nc4perf.c			✓	
tst_opaques.c				✓
tst_parallel3.c			✓	
tst_parallel4.c			✓	
tst_parallel5.c			✓	
tst_parallel.c				✓
tst_put_vars.c				✓
tst_put_vars_two_unlim_dim.c		✓		
tst_rehash.c				✓
tst_rename2.c				✓
tst_rename.c				✓
tst_simplerw_coll_r.c				✓
tst_strings2.c				✓
tst_strings.c				✓
tst_sync.c				✓
tst_types.c				✓
tst_udf.c				✓
tst_unlim_vars.c				✓
tst_utf8.c				✓
tst_utils.c		✓		
tst_v2.c				✓
tst_varms.c				✓
tst_vars2.c				✓
tst_vars3.c				✓
tst_vars4.c				✓
tst_vars.c				✓
tst_vl.c				✓
tst_xplatform2.c			✓	
tst_xplatform.c			✓	
t_type.c				✓

Table 12.3.: List of nc_test4 files – Part III

According to the classification presented in Tables 12.1, 12.2 and 12.3, the 78 **Working** tests were used to demonstrate ESDM functionalities. There are now five categories in which each test was classified according to their status:

Success It means the original test is successful when using ESDM.

Partial Success It means the original test is successful when using ESDM, but some parts of the test code had to be removed because ESDM does not support that specific feature.

Inconclusive It means the original test is not successful when using ESDM, and some features still need to be implemented or revised to provide the expected result. The missing features should be available in the short term.

Failure It means the original test is not successful when using ESDM. Some of the missing features should be available in a medium-term and others are not expected to work with ESDM at all.

Not Tested Yet It means the original test was not tested yet. This category usually represents tests that are too long to run, tests that demand a large amount of memory or processing or tests involving features that are to be implemented.

Tables 12.4 and 12.5 presents the current status of each test. Table 12.6 summarises the results.

Filename	Success	Partial Success	Inconclusive	Failure	Not Tested Yet
cdm_sea_soundings.c				✓	
h5testszip.c	✓				
test_szip.c		✓			
tst_atts.c		✓			
tst_atts1.c		✓			
tst_atts2.c		✓			
tst_atts3.c		✓			
tst_attsperf.c	✓				
tst_atts_string_rewrite.c	✓				
tst_bug324.c		✓			
tst_camrun.c	✓				
tst_chunks.c				✓	
tst_chunks2.c		✓			
tst_chunks3.c					✓
tst_compounds.c				✓	
tst_compounds2.c				✓	
tst_compounds3.c				✓	
tst_converts.c		✓			
tst_converts2.c				✓	
tst_coords.c		✓			
tst_coords2.c		✓			
tst_coords3.c		✓			
tst_create_files.c	✓				
tst_dims.c		✓			
tst_dims2.c		✓			
tst_dims3.c				✓	
tst_elatefill.c				✓	
tst_empty_vlen_unlim.c				✓	
tst_endian_fill.c				✓	
tst_enums.c				✓	
Partial Total	5	13	0	11	1

Table 12.4.: Status – Table I

Filename	Success	Partial Success	Inconclusive	Failure	Not Tested Yet
tst_files.c		✓			
tst_files3.c		✓			
tst_files4.c				✓	
tst_files5.c	✓				
tst_files6.c		✓			
tst_fill_attr_vanish.c		✓			
tst_fillbug.c		✓			
tst_fills.c		✓			
tst_fills2.c				✓	
tst_filterparser.c	✓				
tst_grps.c				✓	
tst_grps2.c				✓	
tst_h5_endians.c				✓	
tst_hdf5_file_compat.c				✓	
tst_h_refs.c				✓	
tst_h_scalar.c				✓	
tst_h_strbug.c				✓	
tst_interops.c				✓	
tst_interops5.c				✓	
tst_interops6.c				✓	
tst_large.c	✓				
tst_large2.c					✓
tst_mem.c	✓				
tst_mode.c		✓			
tst_mpi_parallel.c	✓				
tst_opaques.c				✓	
tst_parallel.c			✓		
tst_put_vars.c	✓				
tst_rehash.c	✓				
tst_rename.c		✓			
tst_rename2.c	✓				
tst_simplerw_coll_r.c			✓		
tst_strings.c				✓	
tst_strings2.c				✓	
tst_sync.c		✓			
tst_types.c				✓	
tst_udf.c				✓	
tst_unlim_vars.c	✓				
tst_utf8.c				✓	
tst_v2.c	✓				
tst_varms.c	✓				
tst_vars.c		✓			
tst_vars2.c		✓			
tst_vars3.c		✓			
tst_vars4.c		✓			
tst_vl.c				✓	
t_type.c	✓				
Partial Total	12	12	2	18	1

Table 12.5.: Status – Table II

Filename	Success	Partial Success	Inconclusive	Failure	Not Tested Yet
Partial Total	5	13	0	11	1
Partial Total	12	12	2	18	1
Total	17	25	2	29	2

Table 12.6.: Status – Tables I and II

12.1. Specific Information For The Tests

This chapter introduces the tests provided by NetCDF in its documentation. For each test, we have four descriptors:

Description The short description provided by the original test.

Output The current output for the test using ESDM.

Comments What was modified in the original test, if any.

Status The current status for the test.

12.2. Test cdm_sea_soundings.c

Description: The cdm tests confirm compliance with the Common Data Model. This file creates some sample data structures to hold sea soundings.

Output: [ESDM NC] WARN ESDM_def_vlen():1889. ESDM does not support user-defined datatypes from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/cdm_sea_soundings.c, line: 45

Comments: ESDM does not support user-defined datatypes from NetCDF!

Status: FAILURE!

12.3. Test h5testszip.c

Description: Example illustrates the use of SZIP compression in HDF5.

Output: ***PASS

Comments: No comments.

Status: SUCCESS!

12.4. Test test_szip.c

Description: Example illustrates the use of SZIP compression in NetCDF5.

Output: *** Tests successful!

Comments: Remove lines 70-88. ESDM does not support compression!

Status: PARTIAL SUCCESS!

12.5. FIX ME! Test tst_atts.c

Description: Test the NetCDF-4 attribute code.

Output: *** Tests successful!

Comments: Remove lines 78-101, 180-186. Access mode.

Remove lines 204, 207. The test works with ESDM.

Remove lines 115-118, 148, 231-269, 280-317, 328-335, 341-343, 353-355, 365-367, 375. Testing invalid parameters as input!

Status: PARTIAL SUCCESS!

12.6. FIX ME! Test tst_atts1.c

Description: Test attributes.

Output: *** Tests successful!

Comments: Remove lines 191-209. No error produced. I cannot verify the output. tst_bug15.

Remove lines 279-619. tst_bug16.

Remove lines 702-786. Testing invalid parameters as input!

Remove lines 815-823. tst_bug45.

Status: PARTIAL SUCCESS!

12.7. FIX ME! Test tst_atts2.c!

Description: Test copy of attributes.

Output: *** Tests successful!

Comments: Remove lines 53-82, 157-259. ESDM does not support user-defined datatypes from NetCDF!

Remove lines 101-105. Testing invalid parameters as input!

Lines 108-119. tst_bug19

Status: PARTIAL SUCCESS!

12.8. Test tst_atts3.c

Description: This is a very simple example which writes a NetCDF file with Unicode names encoded with UTF-8. It is the NETCDF3 equivalent of tst_unicode.c

Output: *** Tests successful!

Comments: Remove lines 2393-2403, 2431-2489. Testing invalid parameters as input!

Status: PARTIAL SUCCESS!

12.9. Test tst_attsperf.c

Description: Test the NetCDF-4 attribute code.

Output: *** Tests successful!

Comments: Too long to run!

Status: SUCCESS!

12.10. Test tst_atts_string_rewrite.c

Description: This test was provided by Jeff Whitaker as an example of a bug, specifically, a segfault when re-writing an NC_CHAR attribute as an NC_STRING attribute.

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

12.11. Test tst_bug324.c

Description: No description.

Output: *** Tests successful!

Comments: Remove line 71. ESDM does not support NetCDF Classic Model!

Status: PARTIAL SUCCESS!

12.12. Test tst_camrun.c

Description: This program writes a data file from the CAM model run.

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

12.13. Test tst_chunks.c

Description: Test netcdf-4 variables.

Output: [ESDM NC] WARN ESDM_def_var_deflate():1985. ESDM does not support compression!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_chunks.c, line: 49

Comments: ESDM does not support compression!

Status: FAILURE!

12.14. Test tst_chunks2.c

Description: Test netcdf-4 chunking.

Output: *** Tests successful!

Comments: Remove lines 106, 102-121, 137-139, 170, 173-175, 208, 211-213, 246-252, 287-292, 327-332, 369-375, 408-414, 447-453. ESDM does not support compression!

Status: PARTIAL SUCCESS!

12.15. FIX ME! Test tst_chunks3.c

Description: Runs benchmarks on different chunking sizes.

Output: Too long to run!

Comments: Remove lines 283-296. ESDM does not support compression!

Status: NOT TESTED!

12.16. Test tst_compounds.c

Description: Test netcdf-4 compound type feature.

Output: [ESDM NC] WARN ESDM_def_compound():1829. ESDM does not support user-defined datatypes from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_compounds.c, line: 52

Comments: ESDM does not support user-defined datatypes from NetCDF!

Status: FAILURE!

12.17. Test tst_compounds2.c

Description: Test netcdf-4 compound type feature.

Output: [ESDM NC] WARN ESDM_def_compound():1829. ESDM does not support user-defined datatypes from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_compounds2.c, line: 61

Comments: ESDM does not support user-defined datatypes from NetCDF!

Status: FAILURE!

12.18. Test tst_compounds3.c

Description: Test netcdf-4 compound type feature, even more.

Output: [ESDM NC] WARN ESDM_def_compound():1829. ESDM does not support user-defined datatypes from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_compounds3.c, line: 77

Comments: ESDM does not support user-defined datatypes from NetCDF!

Status: FAILURE!

12.19. Test tst_converts.c

Description: Test data conversions and fill value handling.

Output: *** Tests successful!

Comments: Remove lines 31-60, 71, 91-100. Format.

Status: PARTIAL SUCCESS!

12.20. Test tst_converts2.c

Description: Test even more data conversions.

Output: Sorry! Unexpected result, ../../libsrcesdm_test/tst_converts2.c, line: 43

Comments: ESDM does not support conversion for variables!

Status: FAILURE!

12.21. Test tst_coords.c

Description: Test netcdf-4 coordinate variables and dimensions.

Output: *** Tests successful!

Comments: Remove lines 92-93, 489, 491, 493, 495, 538, 540, 781, 785, 786, 790, 792. ESDM does not keep the same order for dimids!

Remove lines 115, 122. ESDM does not support compression!

Remove lines 134, 137, 141, 145, 149, 153, 155-156, 158, 161. ESDM does not support NetCDF Classic Model!

Remove line 584-603. ESDM does not support groups from NetCDF!

Remove line 606, 608-648. ESDM does not allow two variables with the same name!

Status: PARTIAL SUCCESS!

12.22. Test tst_coords2.c

Description: Test netcdf-4 coordinate variables and dimensions.

Output: *** Tests successful!

Comments: Remove lines 118-119, 125, 127, 129, 131, 133. ESDM does not keep the same order for dimids!

Remove lines 158, 162-167, 173-179. ESDM does not support groups from NetCDF!

Status: PARTIAL SUCCESS!

12.23. Test tst_coords3.c

Description: Test netcdf-4 coordinate variables and dimensions with an example from the CF conventions.

Output: *** Tests successful!

Comments: Remove lines 136, 138, 140, 144-145, 155-156, 177-178, 186-187, 195-196. ESDM does not keep the same order for dimids!

Status: PARTIAL SUCCESS!

12.24. Test tst_create_files.c

Description: This program creates a test file.

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

12.25. FIX ME! Test tst_dims.c

Description: Test netcdf-4 dimensions.

Output: *** Tests successful!

Comments: Remove lines 104-106, 116, 119-120, 136-138, 141, 150, 164, 217, 226, 261-263, 466. Testing invalid parameters as input!

Remove lines 110, 214. if (nc_enddef(ncid)) ERR. tst_bug46

Remove lines 1102, 1165-1168. ESDM does not support conversion for variables!

Remove lines 1257-1258. ESDM does not keep the same order for dimids!

Remove lines 1315-1357. Creates a new file name (file_in) and tries to open it.

Remove lines 284-295. Problem with rename.

Status: PARTIAL SUCCESS!

12.26. FIX ME! Test tst_dims2.c

Description: Test netcdf-4 dimensions some more.

Output: *** Tests successful!

Comments: Remove lines 58-59. ESDM does not keep the same order for dimids!

Remove lines 325-332. Problem with the conversion of vectors! tst_bug48.c

Remove lines 440-441. Testing invalid parameters as input!

Status: PARTIAL SUCCESS!

12.27. Test tst_dims3.c

Description: Test netcdf-4 dimensions inheritance.

Output: [ESDM NC] WARN ESDM_def_grp():1787. ESDM does not support groups from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_dims3.c, line: 47

Comments: ESDM does not support groups from NetCDF!

Status: FAILURE!

12.28. Test tst_elatefill.c

Description: Test proper elatefill return when fill value is assigned outside of the initial define.

Output: line 41 expecting NC_ELATEFILL but got 0

Comments: Help, Julian!

Status: FAILURE!

12.29. Test tst_empty_vlen_unlim.c

Description: This program excersizes HDF5 variable length array code.

Output: [ESDM NC] WARN ESDM_def_vlen():1889. ESDM does not support user-defined datatypes from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_empty_vlen_unlim.c, line: 63

Comments: ESDM does not support user-defined datatypes from NetCDF!

Status: FAILURE!

12.30. Test tst_endian_fill.c

Description: Create a test file with fill values for a variable of specified endianness.

Output: [ESDM NC] WARN ESDM_def_var_endian():2021. NetCDF Feature not supported with ESDM!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_endian_fill.c, line: 35

Comments: ESDM only supports native endianness!

Status: FAILURE!

12.31. Test tst_enums.c

Description: Test netcdf-4 enum types.

Output: [ESDM NC] WARN ESDM_def_enum():1925. ESDM does not support user-defined datatypes from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_enums.c, line: 47

Comments: ESDM does not support user-defined datatypes from NetCDF!

Status: FAILURE!

12.32. Test tst_files.c

Description: Test netcdf-4 file code.

Output: *** Tests successful!

Comments: Remove line 131. Expected error: NC_ERANGE

Remove line 150. if (nc_put_var_uchar(ncid, varid, uchar_out)) ERR; Problem with the conversion.

Remove line 189. Expected error: NC_ENOTINDEFINE

Remove lines 50-52, 251-269, 279, 285, 291, 303, 309, 410-417, 479-481, 517-529, 534. ESDM does not support NetCDF Classic Model!

Remove lines 336-337. ESDM does not support compression!

Status: PARTIAL SUCCESS!

Help, Julian!

12.33. Test tst_files3.c

Description: This is a benchmark program which tests file writes with compressed data.

Output: *** Tests successful!

Comments: Remove lines 87-88, 215. ESDM does not support compression!

Status: PARTIAL SUCCESS!

12.34. Test tst_files4.c

Description: Test netcdf-4 file from user-reported error. This code based on an ncgen output.

Output: [ESDM NC] WARN ESDM_def_grp():1787. ESDM does not support groups from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_files4.c, line: 66

Comments: ESDM does not support groups from NetCDF!

Status: FAILURE!

12.35. Test tst_files5.c

Description: Test netcdf files a bit.

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

12.36. Test tst_files6.c

Description: Test netcdf files a bit.

Output: *** Tests successful!

Comments: Remove line 86. Testing invalid parameters as input!

Status: PARTIAL SUCCESS!

12.37. Test tst_fill_attr_vanish.c

Description: Based on tst_fillbug.c

Output: *** Tests successful!

Comments: Remove lines 91-97. Expected error: NC_ELATEFILL

Status: PARTIAL SUCCESS!

12.38. Test tst_fillbug.c

Description: Test for a bug that Russ found testing fill values.

Output: *** Tests successful!

Comments: Remove lines 64, 66. Testing invalid parameters as input!

Remove line 83. ESDM does not keep the same order for dimids!

Remove lines 72-74, 84-86. Expected error: NC_FILL_FLOAT

Status: PARTIAL SUCCESS!

12.39. FIX ME! Test tst_fills.c

Description: Create a test file with default fill values for variables of each type.

Output:

Comments: Remove lines 24-86. ESDM does not support datatype NC_STRING!

Status:

12.40. Test tst_fills2.c

Description: Create a test file with default fill values for variables of each type.

Output: Sorry! Unexpected result, ../../libsrcesdm_test/tst_fills2.c, line: 48

Comments: ESDM does not support datatype NC_STRING!

Status: FAILURE!

12.41. Test tst_filterparser.c

Description: No description.

Output: SUCCESS!!

Comments: No comments.

Status: SUCCESS!

12.42. Test tst_grps.c

Description: Test netcdf-4 group code.

Output: [ESDM NC] WARN ESDM_def_grp():1787. ESDM does not support groups from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_grps.c, line: 51

Comments: ESDM does not support groups from NetCDF!

Status: FAILURE!

12.43. Test tst_grps2.c

Description: Test netcdf-4 group code some more.

Output: [ESDM NC] WARN ESDM_def_grp():1787. ESDM does not support groups from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_grps2.c, line: 56

Comments: ESDM does not support groups from NetCDF!

Status: FAILURE!

12.44. Test tst_h5_endians.c

Description: No description.

Output: [ESDM NC] WARN ESDM_def_var_endian():2021. NetCDF Feature not supported with ESDM!

Comments: ESDM does not support HDF5 format!

ESDM only supports native endianness!

Status: FAILURE!

12.45. Test tst_hdf5_file_compat.c

Description: Tests library ability to open files generated by a netcdf instance linked against libhdf5 1.10.0.

Output: [ESDM NC] called ESDM_open:300 ref_hdf5_compat1.nc 65536 0 ref_hdf5_compat1.nc

Sorry! Unexpected result, ../../libsrcesdm_test/tst_hdf5_file_compat.c, line: 42

Comments: ESDM does not support HDF5 format!

Status: FAILURE!

12.46. Test tst_h_refs.c

Description: This program tests fixes for reading NetCDF-4 files that contain datasets with reference data types. The NetCDF-4 library should ignore the datasets and attributes that have reference data types and allow the rest of the file to be accessed.

Output: [ESDM NC] called ESDM_open:300 tst_h_refs.h5 65536 0 tst_h_refs.h5

Sorry! Unexpected result, ../../libsrcesdm_test/tst_h_refs.c, line: 85

Comments: ESDM does not support HDF5 format!

Status: FAILURE!

12.47. Test tst_h_scalar.c

Description: This program tests reading HDF5 files that contain scalar attributes and variables, of both string and numeric datatypes. The NetCDF-4 library should allow access to all of these.

Output: [ESDM NC] called ESDM_open:300 tst_h_scalar.h5 65536 0 tst_h_scalar.h5 Sorry! Unexpected result, ../../libsrcesdm_test/tst_h_scalar.c, line: 272

Comments: ESDM does not support HDF5 format!

Status: FAILURE!

12.48. Test tst_h_strbug.c

Description: This program tests fixes for bugs reported with accessing fixed-length scalar string variables and variable-length scalar string attributes from HDF5 files through the NetCDF-4 API.

Output: [ESDM NC] called ESDM_open:300 tst_h_strbug.h5 65536 0 tst_h_strbug.h5

Sorry! Unexpected result, ../../libsrcesdm_test/tst_h_strbug.c, line: 96

Comments: ESDM does not support HDF5 format!

Status: FAILURE!

12.49. Test tst_interops.c

Description: Test that HDF5 and NetCDF-4 can read and write the same file.

Output: [ESDM NC] called ESDM_open:300 tst_interops.h5 65536 0 tst_interops.h5

Sorry! Unexpected result, ../../libsrcesdm_test/tst_interops.c, line: 126

Comments: ESDM does not support HDF5 format!

Status: FAILURE!

12.50. Test tst_interops5.c

Description: Test that HDF5 and NetCDF-4 can read and write the same file.

Output: [ESDM NC] called ESDM_open:300 tst_interops5.h5 65536 0 tst_interops5.h5

Sorry! Unexpected result, ../../libsrcesdm_test/tst_interops5.c, line: 164

Comments: ESDM does not support HDF5 format!

Status: FAILURE!

12.51. Test tst_interops6.c

Description: Test that HDF5 and NetCDF-4 can read and write the same file.

Output: [ESDM NC] called ESDM_open:300 tst_interops6.h5 65536 0 tst_interops6.h5

Sorry! Unexpected result, ../../libsrcesdm_test/tst_interops6.c, line: 159

Comments: ESDM does not support HDF5 format!

Status: FAILURE!

12.52. Test tst_large.c

Description: Test netcdf-4 large file fill values.

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

12.53. FIX ME! Test tst_large2.c

Description: Test large file problems reported by user.

Output: Too long to run!

Comments:

Status: NOT TESTED!

12.54. Test tst_mem.c

Description: Test internal netcdf-4 file code.

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

12.55. Test tst_mode.c

Description: Test some illegal mode combinations

Output: *** Tests successful!

Comments: Remove lines 26-27, 33-34. Expected error: ESDM does not support compression!

Status: PARTIAL SUCCESS!

12.56. Test tst_mpi_parallel.c

Description: This just exercises MPI file I/O to make sure everything's working properly. If this does not work, netcdf/HDF5 parallel I/O also won't work.

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

12.57. Test tst_opaques.c

Description: Test netcdf-4 opaque types.

Output: [ESDM NC] WARN ESDM_def_opaque():1973. ESDM does not support user-defined datatypes from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_opaques.c, line: 49

Comments: ESDM does not support user-defined datatypes from NetCDF!

Status: FAILURE!

12.58. FIX ME! Test tst_parallel.c

Description: This program tests netcdf-4 parallel I/O.

Output: SUCCESS!

Comments:

Status: INCONCLUSIVE!

12.59. Test tst_put_vars.c

Description: No description.

Output: *** SUCCESS writing example file tst_put_vars.nc!

Comments: No comments.

Status: SUCCESS!

12.60. Test tst_rehash.c

Description: Tests to see if the hashmap is being properly updated.

Output: Tests successful!

Comments: No comments.

Status: SUCCESS!

12.61. Test tst_rename.c

Description: Test renames of vars and dims.

Output: *** Tests successful!

Comments: Remove lines 482-555. ESDM does not support groups from NetCDF!

Status: PARTIAL SUCCESS!

12.62. FIX ME! Test tst_rename2.c

Description: Test more renames of vars and dims.

Output: *** Tests successful!

Comments: Remove lines 52-55. ESDM does not support enddef settings!

```
// if (enddef_setting) {
//   if (nc_enddef(ncid)) ERR;
//   if (nc_redef(ncid)) ERR;
// }
```

Status: PARTIAL SUCCESS!

12.63. FIX ME! Test tst_simplerw_coll_r.c

Description: This test is for parallel IO and the collective access of metadata with HDF5.

Output: Remove lines 342-345.

Comments:

Status: INCONCLUSIVE!

12.64. Test tst_strings.c

Description: Test netcdf-4 string types.

Output: Sorry! Unexpected result, ../../libsrcesdm_test/tst_strings.c, line: 36

Comments: ESDM does not support datatype NC_STRING!

Status: FAILURE!

12.65. Test tst_strings2.c

Description: Test netcdf-4 string types.

Output: Sorry! Unexpected result, ../../libsrcesdm_test/tst_strings2.c, line: 33

Comments: ESDM does not support datatype NC_STRING!

Status: FAILURE!

12.66. Test tst_sync.c

Description: Test netcdf-4 syncs.

Output: *** Tests successful!

Comments: Remove line 128. ESDM does not support NetCDF Classic Model!

Status: PARTIAL SUCCESS!

12.67. Test tst_types.c

Description: Test netcdf-4 types.

Output: [ESDM NC] WARN ESDM_inq_type_equal():1762. ESDM does not support user-defined datatypes from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_types.c, line: 106

Comments: ESDM does not support user-defined datatypes from NetCDF!

Status: FAILURE!

12.68. Test tst_udf.c

Description: Test user-defined formats.

Output: Sorry! Unexpected result, ../../libsrcesdm_test/tst_udf.c, line: 233

Comments: ESDM does not support user-defined datatypes from NetCDF!

```
Thread 1 "tst_udf" hit Breakpoint 1, nc_def_user_format (mode_flag=64, dispatch_table=0x555555758020 <tst_dispatcher>, magic_number=0x0) at ../../libdispatch/dfi
129         if (mode_flag != NC_UDF0 && mode_flag != NC_UDF1)
(gdb) s
131         if (!dispatch_table)
(gdb) n
133         if (magic_number && strlen(magic_number) > NC_MAX_MAGIC_NUMBER_LEN)
(gdb)
138         switch(mode_flag) {
(gdb)
140            UDF0_dispatch_table = dispatch_table;
(gdb)
141            if (magic_number)
(gdb)
143            break;
(gdb)
151         return NC_NOERR;
(gdb)
152     }
```

Status: FAILURE!

Help, Julian!

12.69. Test tst_unlim_vars.c

Description: Test netcdf-4 variables with unlimited dimensions.

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

12.70. Test tst_utf8.c

Description: This is a very simple example which writes a NetCDF file with Unicode names encoded with UTF-8. It is the NETCDF3 equivalent of tst_unicode.c

Output: Sorry! Unexpected result, ../../libsrcesdm_test/tst_utf8.c, line: 181

Comments: ESDM does not support Unicode names encoded with UTF-8!

//TODO Insert in the function that does this test this message.

Status: FAILURE!

Help, Julian!

12.71. Test tst_v2.c

Description: Test internal netcdf-4 file code.

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

12.72. Test tst_varms.c

Description: Test netcdf-4 mapped var operations.

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

12.73. Test tst_vars.c

Description: Test netcdf-4 variables.

Output: *** Tests successful!

Comments: Remove lines 354-369, 859-860. Testing invalid parameters as input!

Remove lines 454-477. ESDM_inq_typeid() NOT IMPLEMENTED

Remove lines 527-564, 567-604, 731-768, 771-808. ESDM does not support conversion for variables!

Remove lines 618:2:638, 834-835, 848-851, 872-875, 914, 930-933, 953-956, 997-998, 1015-1019, 1039-1044, 1087-1091, 1129-1133. ESDM does not support compression!

Remove lines 1292-1310. ESDM does not support NetCDF Classic Model!

//TODO WARN ESDM_inq_typeid():1549 NOT IMPLEMENTED

Status: PARTIAL SUCCESS!

12.74. Test tst_vars2.c

Description: Test netcdf-4 variables.

Output: *** Tests successful!

Comments: Remove lines 93-95. ESDM does not support NetCDF Classic Model!

Remove lines 132-133, 164-165. ESDM_INCOMPLETE_DATA.

Remove line 148. Mode.

Remove line 204, 240, 300, 312, 321, 323, 386, 397, 399, 408, 410. I think it's because the att was not inserted.

Remove lines 434-439, 522-524, 530-535, 541, 544-551, 638-639, 644-646, 652, 716-725, 744-745, 859-860, 1060-1062, 1079-1083, 1107-1111, 1232-1259, 1242-1243, 1274-1275, 1291-1318, 1457-1458. Testing invalid parameters as input!

Remove lines 703, 732-733, 757-758. ESDM only supports native endianness!

Remove lines 767-768, 777-782, 784-1032, 848-851, 914, 1145, 1158-1159, 1179-1180, 1346-1347, 1359-1362, 1378-1381, 1465, 1469-1470, 1473, 1490, 1499, 1505, 1532-1542, 1566-1574.

ESDM does not support compression!

Remove lines 1270, 1407-1426, 1468, 1476-1487, 1493-1496, 1502, 1529. ESDM does not support user-defined datatypes from NetCDF!

Remove lines 1284-1285. ESDM does not support groups from NetCDF!

Remove lines 1286-1287. //TODO Implement function nc_inq_typeids.

Status: PARTIAL SUCCESS!

12.75. Test tst_vars3.c

Description: Test netcdf-4 variables.

Output: *** Tests successful!

Comments: Remove lines 115, 117, 119, 123-124, 126-127, 140, 142, 144, 149-150, 152-153, 285. ESDM does not keep the same order for dimids!

Remove lines 159-197, 219. ESDM only supports native endianness!

Remove lines 229-230, 311, 447. ESDM does not support compression!

Remove lines 270-271. Testing invalid parameters as input!

Remove lines 274-276, 286-288. Expected error: NC_FILL_FLOAT.

Remove lines 322-363. ESDM does not support groups from NetCDF!

Remove lines 409-558. ESDM does not support filters!

Status: PARTIAL SUCCESS!

12.76. Test tst_vars4.c

Description: Test netcdf-4 variables.

Output: *** Tests successful!

Comments: Remove lines 74-76, 81-82, 96-98, 103-104. ESDM does not support compression!

Status: PARTIAL SUCCESS!

12.77. Test tst_vl.c

Description: Test netcdf-4 variable length code.

Output: [ESDM NC] WARN ESDM_def_vlen():1889. ESDM does not support user-defined datatypes from NetCDF!

Sorry! Unexpected result, ../../libsrcesdm_test/tst_vl.c, line: 55

Comments: ESDM does not support user-defined datatypes from NetCDF!

Status: FAILURE!

12.78. Test t_type.c

Description: This test program is only built if NetCDF-4 is disabled. It tests the NetCDF-3 version of NC_inq_type().

Output: *** Tests successful!

Comments: No comments.

Status: SUCCESS!

13. Python Tests

13.1. Introduction

There is a Python module for accessing NetCDF files. We provide a patch to the module to support ESDM. The patch and instructions to run the tests are available in the directory **esdm-netcdf/dev**. The procedures to run the tests are following.

Run the build script:

```
$ ./build.sh
```

Python tests are created now inside the directory **dev/netcdf4-python/test**. Go to that directory.

```
$ cd netcdf4-python/test
```

Now, copy or link the **_esdm.conf** file to this directory. A possible example is:

```
$ cp ../../../build/libsrcesdm_test/_esdm.conf .
```

All Python tests start with the prefix **tst_**. This string has to change to enable us to run **pytest**. To do that, one option is use the the **mmv** utility. Note that the tests names need to be changed, but the tests files (**.nc** files) have to remain unchanged.

The tool can be installed and used in Debian-based distributions as follows:

```
$ sudo apt-get install mmv
```

Now, run the **mmv** in the tests.

```
$ mmv tst.py test#1.py
```

and run the **mkfs.esdm** utility with the following parameters.

```
$ mkfs.esdm –create –remove –ignore-errors -g -c _esdm.conf
```

Finally, install the **pytest** utility

```
$ sudo apt-get install python3-pytest
```

and run it

```
$ pytest –junitxml results.xml
```

The file **results.xml** can now be uploaded by Jenkins wchi provides the results in Table tab.

13.2. Patch

```
diff --git a/include/netCDF4.pxi b/include/netCDF4.pxi
index 625752a..7bcd019 100644
--- a/include/netCDF4.pxi
+++ b/include/netCDF4.pxi
@@ -49,6 +49,7 @@ cdef extern from "netcdf.h":
         NC_WRITE # read & write
         # Use these 'mode' flags for nc_create.
         NC_CLOBBER
+        NC_ESDM # ESDM support for Python
         NC_NOCLOBBER # Don't destroy existing file on create
         NC_64BIT_OFFSET # Use large (64-bit) file offsets
         NC_64BIT_DATA # Use cdf-5 format
@@ -122,6 +123,7 @@ cdef extern from "netcdf.h":
         NC_FORMAT_NC3
         NC_FORMAT_NC_HDF4
         NC_FORMAT_NC_HDF5
+        NC_FORMATX_ESDM
         NC_FORMAT_DAP2
         NC_FORMAT_DAP4
         NC_FORMAT_PNETCDF
@@ -704,7 +706,7 @@ IF HAS_NC_CREATE_MEM:
             int flags
         int nc_close_memio(int ncid, NC_memio* info);

-IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
     cdef extern from "mpi-compat.h": pass
     cdef extern from "netcdf_par.h":
         ctypedef int MPI_Comm
diff --git a/netCDF4/_netCDF4.pyx b/netCDF4/_netCDF4.pyx
index b693354..58d02e8 100644
--- a/netCDF4/_netCDF4.pyx
+++ b/netCDF4/_netCDF4.pyx
@@ -1264,7 +1264,7 @@ import_array()
 include "constants.pyx"
 include "membuf.pyx"
 include "netCDF4.pxi"
-IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
     cimport mpi4py.MPI as MPI
     from mpi4py.libmpi cimport MPI_Comm, MPI_Info, MPI_Comm_dup,
   ↪ MPI_Info_dup, \
                                 MPI_Comm_free, MPI_Info_free, MPI_INFO_NULL
   ↪ ,\
@@ -1344,11 +1344,13 @@ _intnptonctype  = {'i1' : NC_BYTE,
 # create dictionary mapping string identifiers to netcdf format codes
 _format_dict  = {'NETCDF3_CLASSIC' : NC_FORMAT_CLASSIC,
                  'NETCDF4_CLASSIC' : NC_FORMAT_NETCDF4_CLASSIC,
-                 'NETCDF4'         : NC_FORMAT_NETCDF4}
+                 'NETCDF4'         : NC_FORMAT_NETCDF4,
+                 'ESDM'            : NC_FORMATX_ESDM}
 # create dictionary mapping string identifiers to netcdf create format codes
 _cmode_dict  = {'NETCDF3_CLASSIC' : NC_CLASSIC_MODEL,
                 'NETCDF4_CLASSIC' : NC_CLASSIC_MODEL | NC_NETCDF4,
-                'NETCDF4'         : NC_NETCDF4}
+                'NETCDF4'         : NC_NETCDF4,
+                'ESDM'            : NC_ESDM}
 IF HAS_CDF5_FORMAT:
     # NETCDF3_64BIT deprecated, saved for compatibility.
     # use NETCDF3_64BIT_OFFSET instead.
@@ -1547,6 +1549,8 @@ cdef _get_full_format(int grpid):
             return 'DAP2'
         elif formatp == NC_FORMAT_DAP4:
             return 'DAP4'
+        elif formatp == NC_FORMATX_ESDM:
+            return 'ESDM'
         elif formatp == NC_FORMAT_UNDEFINED:
             return 'UNDEFINED'
     ELSE:
@@ -1578,7 +1582,7 @@ be raised in the next release."""
                 PyMem_Free(string_ptrs)
         else:
             # don't allow string array attributes in NETCDF3 files.
-            if is_netcdf3 and N > 1:
+            if is_netcdf3 and N > 1 and fmt != 'ESDM':
                 msg='array string attributes can only be written with NETCDF4'
                 raise IOError(msg)
             if not value_arr.shape:
@@ -1593,7 +1597,7 @@ be raised in the next release."""
     # if array is 64 bit integers or
     # if 64-bit datatype not supported, cast to 32 bit integers.
     fmt = _get_format(grp._grpid)
-    is_netcdf3 = fmt.startswith('NETCDF3') or fmt == 'NETCDF4_CLASSIC'
+    is_netcdf3 = fmt.startswith('NETCDF3') or fmt == 'NETCDF4_CLASSIC' or fmt == 'ESDM'
     if value_arr.dtype.str[1:] == 'i8' and ('i8' not in _supportedtypes or \
       (is_netcdf3 and fmt != 'NETCDF3_64BIT_DATA')):
         value_arr = value_arr.astype('i4')
@@ -2030,7 +2034,7 @@ strings.
     __pdoc__['Dataset.data_model']=\
     """`data_model` describes the netCDF
     data model version, one of `NETCDF3_CLASSIC`, `NETCDF4`,
-    `NETCDF4_CLASSIC`, `NETCDF3_64BIT_OFFSET` or `NETCDF3_64BIT_DATA`."""
+    `NETCDF4_CLASSIC`, `NETCDF3_64BIT_OFFSET` or `NETCDF3_64BIT_DATA` or ESDM."""
     __pdoc__['Dataset.file_format']=\
     """same as `data_model`, retained for backwards compatibility."""
     __pdoc__['Dataset.disk_format']=\
@@ -2084,7 +2088,7 @@ strings.

         **`format`**: underlying file format (one of `'NETCDF4',
         'NETCDF4_CLASSIC', 'NETCDF3_CLASSIC'`, `'NETCDF3_64BIT_OFFSET'` or
-        `'NETCDF3_64BIT_DATA'`.
+        `'NETCDF3_64BIT_DATA'` or ESDM.
         Only relevant if `mode = 'w'` (if `mode = 'r','a'` or `'r+'` the file format
         is automatically detected). Default `'NETCDF4'`, which means the data is
         stored in an HDF5 file, using netCDF 4 API features.  Setting
@@ -2156,7 +2160,7 @@ strings.
         cdef char *path
         cdef char namstring[NC_MAX_NAME+1]
         cdef int cmode
-        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
             cdef MPI_Comm mpicomm
             cdef MPI_Info mpiinfo

@@ -2183,7 +2187,7 @@ strings.
                 msg='parallel mode requires MPI enabled netcdf-c'
                 raise ValueError(msg)
             ELSE:
-                parallel_formats = []
+                parallel_formats = ['ESDM']
                 IF HAS_PARALLEL4_SUPPORT:
                     parallel_formats += ['NETCDF4','NETCDF4_CLASSIC']
                 IF HAS_PNETCDF_SUPPORT:
@@ -2222,7 +2226,7 @@ strings.
             else:
                 if clobber:
                     if parallel:
-                        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+                        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or  HAS_ESDM_SUPPORT:
                             ierr = nc_create_par(path, NC_CLOBBER | cmode,  \
                                    mpicomm, mpiinfo, &grpid)
                         ELSE:
@@ -2272,7 +2276,7 @@ strings.
         version 4.4.1 or higher of the netcdf C lib, and rebuild netcdf4-python."""
                     raise ValueError(msg)
             elif parallel:
-                IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+                IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
                     ierr = nc_open_par(path, NC_NOWRITE | NC_MPIIO, \
                            mpicomm, mpiinfo, &grpid)
                 ELSE:
@@ -2853,7 +2857,7 @@ Use if you need to ensure that a netCDF attribute is created with type
 `NC_STRING` if the file format is `NETCDF4`."""
         cdef nc_type xtype
         xtype=-99
-        if self.data_model != 'NETCDF4':
+        if self.data_model != 'NETCDF4' and self.data_model != 'ESDM':
             msg='file format does not support NC_STRING attributes'
             raise IOError(msg)
         _set_att(self, NC_GLOBAL, name, value, xtype=xtype, force_ncstring=True)
@@ -3435,7 +3435,7 @@ return the group that this `netCDF4.Dimension` is a member of."""
 returns `True` if the `netCDF4.Dimension` instance is unlimited, `False` otherwise."""
         cdef int ierr, n, numunlimdims, ndims, nvars, ngatts, xdimid
         cdef int *unlimdimids
-        if self._data_model == 'NETCDF4':
+        if self._data_model == 'NETCDF4' or self._data_model == "ESDM":
             ierr = nc_inq_unlimdims(self._grpid, &numunlimdims, NULL)
             _ensure_nc_success(ierr)
             if numunlimdims == 0:
@@ -4138,7 +4138,7 @@ Use if you need to ensure that a netCDF attribute is created with type
 Use if you need to set an attribute to an array of variable-length strings."""
         cdef nc_type xtype
         xtype=-99
-        if self._grp.data_model != 'NETCDF4':
+        if self._grp.data_model != 'NETCDF4' and self._grp.data_model != 'ESDM':
             msg='file format does not support NC_STRING attributes'
             raise IOError(msg)
         _set_att(self._grp, self._varid, name, value, xtype=xtype, force_ncstring=True)
diff --git a/setup.py b/setup.py
index febc020..d7ba091 100644
--- a/setup.py
+++ b/setup.py
@@ -58,6 +58,7 @@ def check_api(inc_dirs):
     has_nc_create_mem = False
     has_parallel4_support = False
     has_pnetcdf_support = False
+    has_esdm_support = False

     for d in inc_dirs:
         try:
@@ -564,6 +565,7 @@ if 'sdist' not in sys.argv[1:] and 'clean' not in sys.argv[1:]:
     else:
         sys.stdout.write('netcdf lib does not have pnetcdf parallel functions\n')
         f.write('DEF HAS_PNETCDF_SUPPORT = 0\n')
+    f.write('DEF HAS_ESDM_SUPPORT = 1\n') # TODO Fixme

     f.close()
```

```
+        NC_ESDM # ESDM support for Python
+        NC_FORMATX_ESDM
-IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
-IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
-                 'NETCDF4'         : NC_FORMAT_NETCDF4}
+                 'NETCDF4'         : NC_FORMAT_NETCDF4,
+                 'ESDM'            : NC_FORMATX_ESDM}
-                'NETCDF4'         : NC_NETCDF4}
+                'NETCDF4'         : NC_NETCDF4,
+                'ESDM'            : NC_ESDM}
+        elif formatp == NC_FORMATX_ESDM:
+            return 'ESDM'
-            if is_netcdf3 and N > 1:
+            if is_netcdf3 and N > 1 and fmt != 'ESDM':
-    is_netcdf3 = fmt.startswith('NETCDF3') or fmt == 'NETCDF4_CLASSIC'
+    is_netcdf3 = fmt.startswith('NETCDF3') or fmt == 'NETCDF4_CLASSIC' or
   ↪ fmt == 'ESDM'
-    `NETCDF4_CLASSIC`, `NETCDF3_64BIT_OFFSET` or `NETCDF3_64BIT_DATA`."""
+    `NETCDF4_CLASSIC`, `NETCDF3_64BIT_OFFSET` or `NETCDF3_64BIT_DATA` or
   ↪ ESDM."""
-        `'NETCDF3_64BIT_DATA'`.
+        `'NETCDF3_64BIT_DATA'` or ESDM.
-        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
   ↪ HAS_ESDM_SUPPORT:
-                parallel_formats = []
+                parallel_formats = ['ESDM']
-                        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+                        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
   ↪  HAS_ESDM_SUPPORT:
-                IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT:
+                IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
   ↪ HAS_ESDM_SUPPORT:
-        if self.data_model != 'NETCDF4':
+        if self.data_model != 'NETCDF4' and self.data_model != 'ESDM':
-        if self._data_model == 'NETCDF4':
+        if self._data_model == 'NETCDF4' or self._data_model == "ESDM":
-        if self._grp.data_model != 'NETCDF4':
+        if self._grp.data_model != 'NETCDF4' and self._grp.data_model != '
   ↪ ESDM':
+    has_esdm_support = False
+    f.write('DEF HAS_ESDM_SUPPORT = 1\n') # TODO Fixme
```

```
+        NC_ESDM # ESDM support for Python
+        NC_FORMATX_ESDM
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
+IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or HAS_ESDM_SUPPORT:
+                 'NETCDF4'         : NC_FORMAT_NETCDF4,
+                 'ESDM'            : NC_FORMATX_ESDM}
+                'NETCDF4'         : NC_NETCDF4,
+                'ESDM'            : NC_ESDM}
+        elif formatp == NC_FORMATX_ESDM:
+            return 'ESDM'
+            if is_netcdf3 and N > 1 and fmt != 'ESDM':
+    is_netcdf3 = fmt.startswith('NETCDF3') or fmt == 'NETCDF4_CLASSIC' or
   ↪ fmt == 'ESDM'
+    `NETCDF4_CLASSIC`, `NETCDF3_64BIT_OFFSET` or `NETCDF3_64BIT_DATA` or
   ↪ ESDM."""
+        `'NETCDF3_64BIT_DATA'` or ESDM.
+        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
   ↪ HAS_ESDM_SUPPORT:
+                parallel_formats = ['ESDM']
+                        IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
   ↪  HAS_ESDM_SUPPORT:
+                IF HAS_PARALLEL4_SUPPORT or HAS_PNETCDF_SUPPORT or
   ↪ HAS_ESDM_SUPPORT:
+        if self.data_model != 'NETCDF4' and self.data_model != 'ESDM':
+        if self._data_model == 'NETCDF4' or self._data_model == "ESDM":
+        if self._grp.data_model != 'NETCDF4' and self._grp.data_model != '
   ↪ ESDM':
+    has_esdm_support = False
+    f.write('DEF HAS_ESDM_SUPPORT = 1\n')
```

14. NetCDF Benchmark

14.1. Introduction

So far, we tested individual features of ESDM, as part of this section we discuss the usage of a benchmark application that uses NetCDF similarly to a real application.

The NetCDF Performance Benchmark Tool (NetCDF-Bench) was developed to measure NetCDF performance on devices ranging from notebooks to large HPC systems. It mimics the typical I/O behavior of scientific climate applications and captures the performance on each node/process. In the end, it aggregates the data to human readable summary[1].

Scripts to setup the NetCDF Benchmark with ESDM can be found in the directory

```
esiwace/esdm-netcdf/libsrcesdm_test/netcdf-bench
```

The procedures to run the tests are following.

Run the prepare script:

```
./prepare.sh
```

Copy or link the **_esdm.conf** file to this directory. A possible example is:

```
$ cp ../../../build/libsrcesdm_test/_esdm.conf .
```

Now, link the **_esdm.conf** file to the **esdm.conf** file.

```
ln -s _esdm.conf esdm.conf
```

Run the **mkfs.esdm** utility with the following parameters.

```
$ mkfs.esdm –create –remove –ignore-errors -g -c _esdm.conf
```

[1] https://github.com/joobog/netcdf-bench

The benchmark file is called **benchtool** and it can be run with several parameters. For more information about this benckmark, run the command

```
./benchtool –help
```

14.2. Tests

The benchmark can run with various levels of parallelism, domain decomposition and access patterns.

14.2.1. Test 1

./benchtool -f=esdm://longtest -w

14.2.2. Test 2

./benchtool -f=esdm://longtest -r

14.2.3. Test 3

./benchtool -f=esdm://longtest -w -r

14.2.4. Test 4

mpiexec -np 2 ./src/benchtool -f="esdm://test.esdm" -n=1 -p=2

14.2.5. Test 5

```
./benchtool -f=esdm://testfile -F=1

Benchtool (datatype: int)
DEBUG [0]          main.c:364   dgeom (100:100:100:10)
DEBUG [0]          main.c:365   bgeom (1:100:100:10)
DEBUG [0]          main.c:369   (nn 1, ppn 1)
DEBUG [0]          main.c:370   test filename esdm://testfile
Data geometry (t:x:y:z x sizeof(type))     100:100:100:10 x 4 bytes
Block geometry (t:x:y:z x sizeof(type))      1:100:100:10 x 4 bytes
Datasize                                             40000000 bytes                 (40.0 MB)
Blocksize                                              400000 bytes                 (400.0 kB)
I/O Access                                        independent
Storage                                            contiguous
File length                                             fixed
Fill value                                                yes
0 = 100
1 = 100
2 = 100
3 = 10
                                                                               min                  avg                   max
benchmark:write      Open time                                        0.0018334580         0.0018334580         0.0018334580 secs
benchmark:write      I/O time                                         0.0652808010         0.0652808010         0.0652808010 secs
benchmark:write      Close time                                       0.0004117420         0.0004117420         0.0004117420 secs
benchmark:write      I/O Performance (w/o open/close)               584.3520923750       584.3520923750       584.3520923750 MiB/s
benchmark:write      I/O Performance                                564.9227274190       564.9227274190       564.9227274190 MiB/s
```

```
DEBUG [0]        report.c:362   REPORT_END

ESDM has not been shutdown correctly. Stacktrace:
3: /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xee) [0x7f1a4b837b9e]
4: ./benchtool(_start+0x2a) [0x55b91b68ce6a]
```

14.2.6. Test 6

```
./benchtool -f=esdm://testfile

Benchtool (datatype: int)
Data geometry (t:x:y:z x sizeof(type))     100:100:100:10 x 4 bytes
Block geometry (t:x:y:z x sizeof(type))      1:100:100:10 x 4 bytes
Datasize                                         40000000 bytes               (40.0 MB)
Blocksize                                          400000 bytes               (400.0 kB)
I/O Access                                    independent
Storage                                        contiguous
File length                                         fixed
File value                                             no
DEBUG [0]     benchmark.c:213   OPEN_BENCHMARK
0 = 100
1 = 100
2 = 100
3 = 10
                                                                        min                 avg                 max
benchmark:write      Open time                                 0.0017575690        0.0017575690        0.0017575690 secs
benchmark:write      I/O time                                  0.0612307650        0.0612307650        0.0612307650 secs
benchmark:write      Close time                                0.0006145310        0.0006145310        0.0006145310 secs
benchmark:write      I/O Performance (w/o open/close)        623.0033653140      623.0033653140      623.0033653140 MiB/s
benchmark:write      I/O Performance                         599.7681496880      599.7681496880      599.7681496880 MiB/s
DEBUG [0]        report.c:362   REPORT_END

ESDM has not been shutdown correctly. Stacktrace:
3: /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xee) [0x7f29b95bab9e]
4: ./benchtool(_start+0x2a) [0x556502704e6a]
```

14.2.7. Test 7

```
./benchtool -f=esdm://longtest -w -r

Benchtool (datatype: int)
Data geometry (t:x:y:z x sizeof(type))     100:100:100:10 x 4 bytes
Block geometry (t:x:y:z x sizeof(type))      1:100:100:10 x 4 bytes
Datasize                                         40000000 bytes               (40.0 MB)
Blocksize                                          400000 bytes               (400.0 kB)
I/O Access                                    independent
Storage                                        contiguous
File length                                         fixed
File value                                             no
DEBUG [0]     benchmark.c:213   OPEN_BENCHMARK
0 = 100
1 = 100
2 = 100
3 = 10
                                                                        min                 avg                 max
benchmark:write      Open time                                 0.0021035570        0.0021035570        0.0021035570 secs
benchmark:write      I/O time                                  0.0550000690        0.0550000690        0.0550000690 secs
benchmark:write      Close time                                0.0005124810        0.0005124810        0.0005124810 secs
benchmark:write      I/O Performance (w/o open/close)        693.5804508943      693.5804508943      693.5804508943 MiB/s
benchmark:write      I/O Performance                         662.0886873884      662.0886873884      662.0886873884 MiB/s
                                                                        min                 avg                 max
benchmark:read       Open time                                 0.0010185700        0.0010185700        0.0010185700 secs
benchmark:read       I/O time                                  0.0209648240        0.0209648240        0.0209648240 secs
benchmark:read       Close time                                0.0003341830        0.0003341830        0.0003341830 secs
benchmark:read       I/O Performance (w/o open/close)       1819.5703744858     1819.5703744858     1819.5703744858 MiB/s
benchmark:read       I/O Performance                        1709.2793117792     1709.2793117792     1709.2793117792 MiB/s
DEBUG [0]        report.c:362   REPORT_END

ESDM has not been shutdown correctly. Stacktrace:
3: /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xee) [0x7f71e0849b9e]
4: ./benchtool(_start+0x2a) [0x5611f27a1e6a]
```

14.2.8. Test 8

```
./run-test.sh

#!/bin/bash

./benchtool -f=esdm://longtest -w -r

mpiexec -np 2 ./src/benchtool -f="esdm://test.esdm" -n=1 -p=2

echo "Cleanup"
```

```
rm -rf _metadummy _esdm

echo "[OK]"

Benchtool (datatype: int)
Data geometry (t:x:y:z x sizeof(type))     100:100:100:10 x 4 bytes
Block geometry (t:x:y:z x sizeof(type))      1:100:100:10 x 4 bytes
Datasize                                        40000000 bytes                (40.0 MB)
Blocksize                                         400000 bytes                (400.0 kB)
I/O Access                                   independent
Storage                                       contiguous
File length                                        fixed
File value                                            no
DEBUG [0]     benchmark.c:213   OPEN_BENCHMARK
0 = 100
1 = 100
2 = 100
3 = 10
                                                                        min                 avg                 max
benchmark:write      Open time                                 0.0020030390        0.0020030390        0.0020030390 secs
benchmark:write      I/O time                                  0.0648872400        0.0648872400        0.0648872400 secs
benchmark:write      Close time                                0.0003753280        0.0003753280        0.0003753280 secs
benchmark:write      I/O Performance (w/o open/close)        587.8963669180      587.8963669180      587.8963669180 MiB/s
benchmark:write      I/O Performance                         567.1096174930      567.1096174930      567.1096174930 MiB/s
                                                                        min                 avg                 max
benchmark:read       Open time                                 0.0008915110        0.0008915110        0.0008915110 secs
benchmark:read       I/O time                                  0.0168951110        0.0168951110        0.0168951110 secs
benchmark:read       Close time                                0.0004516290        0.0004516290        0.0004516290 secs
benchmark:read       I/O Performance (w/o open/close)       2257.8704962897     2257.8704962897     2257.8704962897 MiB/s
benchmark:read       I/O Performance                        2091.5916039082     2091.5916039082     2091.5916039082 MiB/s
DEBUG [0]        report.c:362   REPORT_END

ESDM has not been shutdown correctly. Stacktrace:
3: /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xee) [0x7f8f101e3b9e]
4: ./benchtool(_start+0x2a) [0x56250989ee6a]
--------------------------------------------------------------------------
mpiexec was unable to launch the specified application as it could not access
or execute an executable:

Executable: ./src/benchtool
Node: lucy

while attempting to start process rank 0.
--------------------------------------------------------------------------
2 total processes failed to start
Cleanup
[OK]
```

15. Tests with nccopy

15.1. Introduction

The nccopy utility copies an input NetCDF file to an output NetCDF file, in any of the four format variants, if possible.[1]

The nccopy utility can be found in the directory

```
esdm-netcdf/build/ncdump/nccopy
```

and the simplest way to call is

```
nccopy input_file output_file
```

As we want to test if the files generated by ESDM are compatible with NetCDF, we are going to run four different tests, described in the next sections. To call ESDM, just insert **esdm://** before the file.

Before start the testing, it is also good to run the **mkfs.esdm** utility with the following parameters.

```
$ mkfs.esdm –create –remove –ignore-errors -g -c _esdm.conf
```

Two files are being tested: smallfile.nc and bigfile.nc. These files are originally NetCDF files. Table 15.1 has information about these files[2].

netcdf smallfile dimensions: lat = 1 ; variables: float lat(lat) ; float lon(lat) ; int64 Elevation(lat, lat) ;

NetCDF File	Size	
sresa1b_ncar_ccsm3-example.nc	2.8 MB	From the Community Climate Sy
test_echam_spectral.nc	281.4 MB	Example model output from the ECHAM general circulatio

Table 15.1.: Sample files following CF conventions.

[1]http://www.doc.ic.ac.uk/~rn710/Installs/netcdf-4.2/man4/html/guide_nccopy.html
[2]https://www.unidata.ucar.edu/software/netcdf/examples/files.html

15.2. Tests

Copy the original NetCDF file to ESDM:

```
nccopy test_echam_spectral.nc esdm:
test_echam_spectral_out.nc
```

Copy the ESDM file to NetCDF format:

```
nccopy esdm:
test_echam_spectral_out.nc test_echam_spectral_final.nc
```

Compare the initial and final files:

```
diff test_echam_spectral.nc test_echam_spectral_final.nc
```

All tests run without output.

```
lucy@lucy:~/esiwace/esdm-netcdf/build/ncdump$ ncdump smallfile.nc
netcdf smallfile {
dimensions:
lat = 1 ;
variables:
float lat(lat) ;
float lon(lat) ;
int64 Elevation(lat, lat) ;
data:

 lat = _ ;

 lon = _ ;

 Elevation =
  _ ;
}

ESDM has not been shutdown correctly. Stacktrace:
3: ncdump(+0x86f9) [0x562111c426f9]
4: /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xe7) [0x7f046adb9b97]
5: ncdump(+0x264a) [0x562111c3c64a]

lucy@lucy:~/esiwace/esdm-netcdf/build/ncdump$ .libs/nccopy smallfile.nc esdm://smallfile_out.esdm
0 = 1
0 = 1
0 = 1
0 = 1

ESDM has not been shutdown correctly. Stacktrace:
3: .libs/nccopy(+0x8c18) [0x5649b5b55c18]
4: /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xe7) [0x7f7da6109b97]
5: .libs/nccopy(+0x29ea) [0x5649b5b4f9ea]

lucy@lucy:~/esiwace/esdm-netcdf/build/ncdump/_metadummy/containers$ more smallfile_out.esdm.md
{"Variables":{"type":"o","data":null,"childs":{"_nc_dims":{"type":"q1@m","data":["lat"]},"_nc_sizes":{"type":"q1@i","da
ta":[1]}}},"dsets":[{"name":"lat","id":"eUyIKbbzj9YOkEpk"},
{"name":"lon","id":"eUyIKbxOPzPXyKqL"},
{"name":"Elevation","id":"eUyIKbrJHZSAAnbt"}]}

lucy@lucy:~/esiwace/esdm-netcdf/build/ncdump/_metadummy/containers$ python -m json.tool smallfile_out.esdm.md
{
    "Variables": {
        "childs": {
            "_nc_dims": {
                "data": [
                    "lat"
                ],
                "type": "q1@m"
            },
            "_nc_sizes": {
                "data": [
                    1
                ],
                "type": "q1@i"
            }
        },
        "data": null,
        "type": "o"
    },
    "dsets": [
        {
            "id": "eUyIKbbzj9YOkEpk",
            "name": "lat"
        },
        {
            "id": "eUyIKbxOPzPXyKqL",
            "name": "lon"
        },
        {
            "id": "eUyIKbrJHZSAAnbt",
            "name": "Elevation"
        }
    ]
}
```

```
lucy@lucy:~/esiwace/esdm-netcdf/build/ncdump$ .libs/nccopy bigfile.nc esdm://bigfile_out.esdm

ESDM has not been shutdown correctly. Stacktrace:
3: .libs/nccopy(+0x8c18) [0x55a117c9fc18]
4: /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xe7) [0x7f6f430a4b97]
5: .libs/nccopy(+0x29ea) [0x55a117c999ea]
```

nccopy esdm://input_file output_file

16. Jenkins

17. Conclusion

A. Conversion

```
Final Results - INT8 to FLOAT

********************************************
INT8_MAX = 127
Maximum FLOAT = 126.000000
Maximum INT8 = 126
********************************************

********************************************
INT8_MIN = -128
Minimum FLOAT = -127.000000
Minimum INT8 = -127
********************************************

Final Results - INT16 to FLOAT

********************************************
INT16_MAX = 32767
Maximum FLOAT = 32766.000000
Maximum INT16 = 32766
********************************************

********************************************
INT16_MIN = -32768
Minimum FLOAT = -32767.000000
Minimum INT16 = -32767
********************************************

Final Results - INT32 to FLOAT

********************************************
INT32_MAX = 2147483647
Maximum FLOAT = 2147483520.000000
Maximum INT32 = 2147483583
********************************************

********************************************
INT32_MIN = -2147483648
Minimum FLOAT = -2147483520.000000
Maximum INT32 = -2147483583
********************************************

Final Results - INT64 to FLOAT

********************************************
INT64_MAX = 9223372036854775807
Maximum FLOAT = 9223371487098961920.000000
Maximum INT64 = 9223371761976865807
********************************************

********************************************
INT64_MIN = -9223372036854775808
Minimum FLOAT = -9223371487098961920.000000
Minimum INT64 = -9223371761976865808
********************************************

Final Results - UINT8 to FLOAT

********************************************
UINT8_MAX = 255
Maximum FLOAT = 254.000000
Maximum UINT8 = 254
********************************************

Final Results - UINT16 to FLOAT

********************************************
UINT16_MAX = 65535
Maximum FLOAT = 65534.000000
Maximum UINT16 = 65534
********************************************

Final Results - UINT32 to FLOAT

********************************************
UINT32_MAX = 4294967295
Maximum FLOAT = 4294967040.000000
Maximum UINT32 = 4294967167
********************************************

Final Results - UINT64 to FLOAT

********************************************
```

```
UINT64_MAX = 18446744073709551615
Maximum FLOAT = 18446742974197923840.000000
Maximum UINT64 = 18446743523953731615
*********************************************

Final Results - INT8 to DOUBLE

*********************************************
INT8_MAX = 127
Maximum DOUBLE = 126.000000
Maximum INT8 = 126
*********************************************

*********************************************
INT8_MIN = -128
Minimum DOUBLE = -127.000000
Minimum INT8 = -127
*********************************************

Final Results - INT16 to DOUBLE

*********************************************
INT16_MAX = 32767
Maximum DOUBLE = 32766.000000
Maximum INT16 = 32766
*********************************************

*********************************************
INT16_MIN = -32768
Minimum DOUBLE = -32767.000000
Minimum INT16 = -32767
*********************************************

Final Results - INT32 to DOUBLE

*********************************************
INT32_MAX = 2147483647
Maximum DOUBLE = 2147483646.000000
Maximum INT32 = 2147483646
*********************************************

*********************************************
INT32_MIN = -2147483648
Minimum DOUBLE = -2147483647.000000
Minimum INT32 = -2147483647
*********************************************

Final Results - INT64 to DOUBLE

*********************************************
INT64_MAX = 9223372036854775807
Maximum DOUBLE = 9223372036854774784.000000
Maximum INT64 = 9223372036854775295
*********************************************

*********************************************
INT64_MIN = -9223372036854775808
Minimum DOUBLE = -9223372036854774784.000000
Minimum INT64 = -9223372036854775295
*********************************************

Final Results - UINT8 to DOUBLE

*********************************************
UINT8_MAX = 255
Maximum DOUBLE = 254.000000
Maximum UINT8 = 254
*********************************************

Final Results - UINT16 to DOUBLE

*********************************************
UINT16_MAX = 65535
Maximum DOUBLE = 65534.000000
Maximum UINT16 = 65534
*********************************************

Final Results - UINT32 to DOUBLE

*********************************************
UINT32_MAX = 4294967295
Maximum DOUBLE = 4294967294.000000
Maximum UINT32 = 4294967294
*********************************************

Final Results - UINT64 to DOUBLE

*********************************************
UINT64_MAX = 18446744073709551615
Maximum DOUBLE = 18446744073709549568.000000
Maximum UINT64 = 18446744073709550591
*********************************************
```