

WRF lateral boundary conditions

(Documented by S.G.Gopalakrishnan, NCEP)

The following topics are discussed in this section:

- (1) Types of boundary conditions
- (2) writing a wrfbdy file in real_XXX.F
- (3) reading wrfbdy file
- (4) Using real boundary conditions

(1) Types of boundary conditions

Several lateral boundary condition (LBC) options are available within the WRF modeling infrastructure. Depending on the type of compile (see WRFV1/README_test_cases or execute ./compile for details) as well as a relevant selection of the type of LBCs from the namelist file, one might be able to use a wide variety of these options within the infrastructure. Broadly, "ideal" and "real" cases for boundary conditions are delineated at the time of compile. While this section explains, briefly, the "ideal" boundary conditions, following sections will describes "real" boundary conditions (i.e., The "specified" option in the namelist). ./share/module_bc.F and ./dyn_nmm/module_BNDRY_COND.F contains all higher-level routines related to the LBCs (Note that module_BNDRY_COND.F is specific to the NMM core.) We focus on how the LBC data is written, stored and finally used in any dynamical core. Note that all subroutines in these modules may be directly called from the solver.

Four types of LBCs are possible within WRF, namely, symmetric, open, periodic and specified. Specified LBCs are frequently used with real data cases, and will be discussed in detail later in this text. A combination, for instance, open for

North-South boundary and cyclic for West-East boundary is also possible and may be particularly useful for ideal cases. Each of these cases is identified within `module_bc.F` using `model` layer, F90 derived data type defined in `module_configure.F`. Note that since `module_bc.F` is a model layer routine, direct use of `module_configure` is possible (i.e., there is no need of using CALL “get_” routines) in `module_bc.F`.

Apart from `module_configure`, `module_bc` uses `module_wrf_error` for producing warnings and errors for an inconsistent LBC configuration. This module (`module_bc.F`) first defines `bdyzone` which is actually the number of points on either side outside of the domain boundary (currently this hardwired to 3). Also, `bdyzone_x` and `bdyzone_y`, which are used to configure domains are set here (currently this hardwired to 3).

Based on the configuration flag, `bdyzone` and the grid id, the first routine in `module_bc.F`, namely, SUBROUTINE `boundary_condition_check` checks the LBC logicals to ensure that the boundary conditions are not over or under specified. The routine also checks that the boundary zone is sufficiently sized for the specified boundary conditions. Either `wrf_message` or `wrf_error_fatal` is called in case of wrong specification of the LBCs in the namelist file.

The following SUBROUTINE `set_physical_bc2d` and SUBROUTINE `set_physical_bc3d` are used to set the data in the boundary region, by direct assignment of input data from some part of the domain, for periodic, symmetric (wall) and open boundary conditions used for ideal cases. Input data to these subroutines may be any prognostic, state variables that are needed to be updated at the boundaries (e.g., `u`, `v`, `t`, `mu`, etc.) Further, depending on the character `variable_in`, which are used to define staggering, and configuration flag (and of course the `bdyzone` that is common to all routines contained in `module_bc.F`), the data is specified or can be overwritten at the boundaries. The example provided here shows how 3d data from the eastern boundary zone (i.e., centered around `ide`) is transferred to the western boundary zone (i.e., centered around `ids`) in the case of periodic boundary conditions along `x` direction. Note

that because cyclic conditions requires communication outside patch, the patch must cover entire range in periodic direction. Also, jstag is set to -1 here.

```
periodicity_x: IF( ( config_flags%periodic_x ) .and. ( ids == ips ) .and. ( ide == ipe ) )THEN

    IF ( its == ids ) THEN

        DO j = MAX(jds,jts-1), MIN(jte+1,jde+jstag)

            DO k = kts, k_end

                DO i = 0,-(bdyzone-1),-1

                    data(ids+i -1, k, j) = data(ide+i -1, k, j)

                ENDDO

            ENDDO

        ENDDO

    ENDIF

.....

.....

ENDIF periodicity_x
```

The following, simplified, example shows how 2D data from the "mass" and "u" points on a staggered "C" grid are copied from an inner grid point (penultimate grid points) of computation onto the eastern boundary. Note that jstag is set to -1 here.

```
open_xe: IF(config_flags%open_xe .and. ite == ide)THEN

    IF ( variable /= 'u' .and. variable /= 'x') THEN

        DO j = MAX(jds,jts-1), MIN(jte+1,jde+jstag)

            data(ide,j) = data(ide-1,j)

        ENDDO

    ELSE

        DO j = MAX(jds,jts-1), MIN(jte+1,jde+jstag)
```

```
        data(ide+1,j) = data(ide,j)

        ENDDO

    END IF

END IF open_xe
```

Subroutines `zero_grad_bdy`, `flow_dep_bdy`, `relax_bdytend`, `spec_bdytend`, `spec_bdyupdate` in `./share/module_bc.F` and all routines in `./dyn_nmm/module_BNDRY_COND.F` are specific to real boundary conditions. SUBROUTINE `zero_grad_bdy` in `./share/module_bc.F` sets zero gradient conditions in the boundary specified region. `spec_zone` (namelist variable) is the width of the outer specified boundary. Other arguments required for this routine are input data, character variable `in` (for example 't' for potential temperature perturbation in the mass core), which are used to define staggering (variables, `istag` and `jstag`), and configuration flag (and of course the `bdyzone` that is common to all routines contained in `module_bc.F`). This routine is currently used in the mass/height core for prescribing W at the boundaries.

The following routine SUBROUTINE `flow_dep_bdy` sets zero gradient conditions for outflow and zero value for inflow in the boundary specified region. Along with the input data (specifically a scalar in this case), configuration flag and `spec_zone`, since the input field that will eventually be updated at the lateral boundaries are scalar, unstaggered fields, the other inputs to these subroutine, namely, velocities, `u` and `v`, will only be used to check their sign. Note that this routine does not use `variable_in` as input argument for the above reason. The SUBROUTINE `spec_bdyupdate` (in `./share/module_bc.F`) uses tendencies of variables computed on longer time steps using external data source as an input (discussed in the subsequent section), along with small time step, `spec_zone`, `variable_in`, and `config_flags`, updates a given input field at the boundary. This routine, too, is used in NCAR core.

The routines provided below, in general, are used within WRF for updating LBCs using external data for real cases.

- (i) relax_bdytend (./share/module_bc.F)
- (ii) spec_bdytend (./share/module_bc.F)
- (iii) BOCOH (./dyn_nmm/ MODULE_BNDRY_COND.F)
- (iv) BOCOV (./dyn_nmm/ MODULE_BNDRY_COND.F)

While the first two routines are used to update LBCs for a “c” grid configuration, the last two are used in the E-grid, NMM core. The way each of the core treats LBCs for real cases may be different and will not be discussed here. However, the ultimate aim is to modify or (re)write data at the boundary for a given field on the basis of a larger scale model forecast (external data). For this purpose both, a given field at the boundary as well as its tendency between a fixed time period are required. Once the external data (both the field as well as tendency) is interpolated for a specific grid configuration (currently wrfsi is used for this purpose) and tendencies of these interpolated meteorological fields are computed from real_XXX.F, then WRF modeling infrastructure reads these interpolated data and writes it onto a file called “wrfbdy_dxx” (xx denotes domain id). Although the outputs from most of the "specified" routines are data either in two or three dimensions, special four dimensional arrays are used to store and retrieve data.

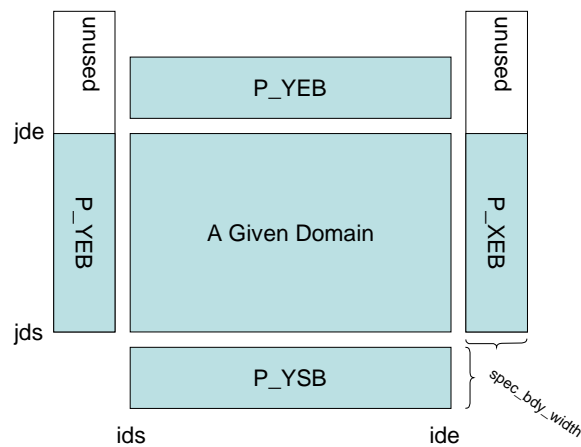
Obviously, all the routines have these four dimensional data, say, field_bdy, field_bdy_tend as one of the arguments. The next section explains how LBCs are written onto a "wrfbdy" input data file and used in the model.

(2) writing a wrfbdy file in real_XXX.F

The lateral boundaries for real cases are stored as, and read from special four dimensional arrays. These arrays are specified in the Registry as follows: the state variable, say for instance U, has its boundary values stored in U_b and the boundary tendency in U_bt. The “_b” and “_bt” arrays are four dimensional (max(ide,jde),kde,spec_bdy_width,4). As indicated by the fourth dimensional

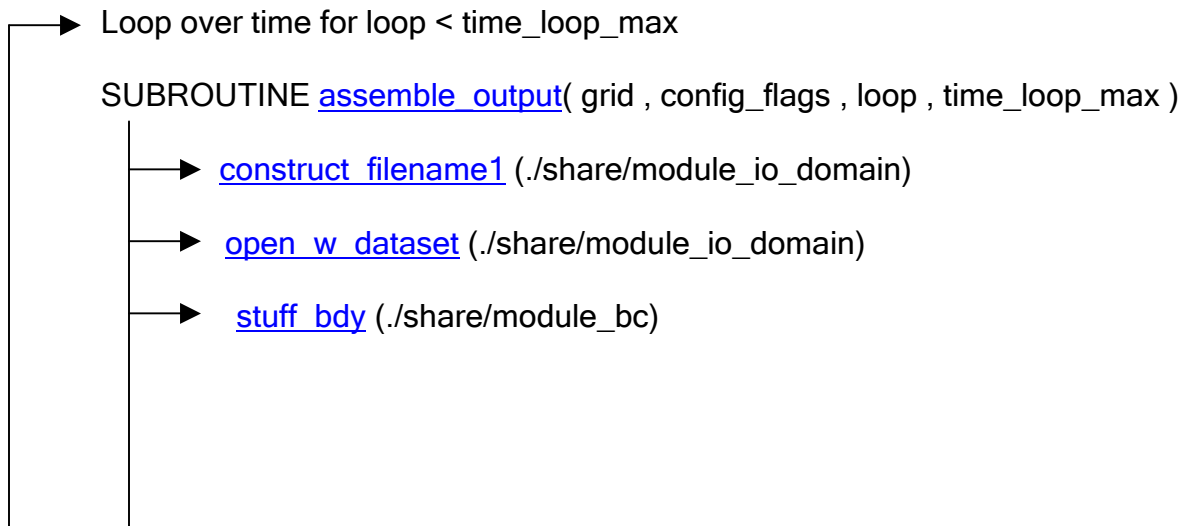
index, the data is stored in the form of four strips, P_XSB (western), P_XEB (eastern), P_YSB (southern) and P_YEB (northern), each denoting a boundary of specified width, `spec_bdy_width` that is controlled by the namelist file. While the second dimension index of these array denotes the maximum vertical dimension, the first index denotes maximum of x and y dimension. Figure below illustrates this array for a case where `ide > jde`

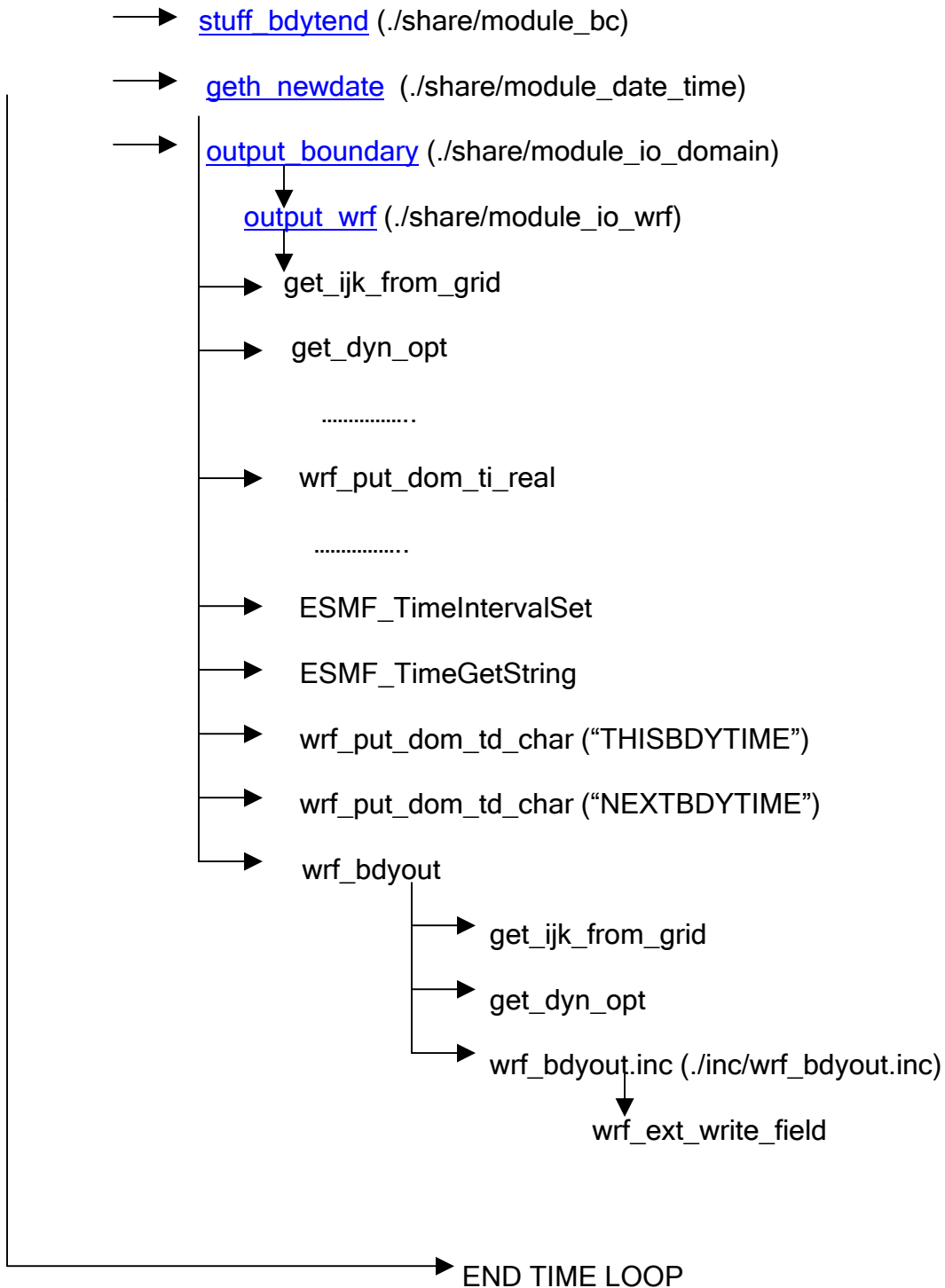
LBC Arrays



The following tree sketches the sequence of how a `wrfbdy` file is written within WRF. This sequence starts off with the real code at the highest level and assumes that grid contains the needed data for a given time.

`real_XXX.F`





Given the boundary file string, i.e., “wrfbdy” [construct filename1](#) returns wrfbdy_d0N, where N is the domain. For the headgrid, [construct filename1](#) returns wrfbdy_d01. [open w dataset](#) is high level package-independent WRF/IO routine that is used to open the constructed file (say, wrfbdy_d01). Specifically,

this is done in two steps by first calling `wrf_open_for_write_begin` and then calling `wrf_open_for_write_commit` (details of this io pair may be explained elsewhere). At this stage, a boundary file, `wrfbdy_d01`, is opened for writing. The user may specify which output format they wish to write in the WRF namelist by setting the integer `io_form`.

Subroutine [stuff_bdy](#) and [stuff_bdytend](#) are called from `real_xxx.F` in main subdirectory and these routines are found in `./share/module_bc.F`. Depending on the staggering, input data that are read in from an external source (for example, `wrf_real_input**` files from SI) is simply redistributed as a four dimensional array, structure of which was explained above (how the data is read in from external source in `real_xxx.F` may be discussed, elsewhere). Note that the LBC arrays are globally dimensioned, yet, as they are not fully dimensioned, still scalable in memory, preserves global address space for dealing with LBCs, and consequently, makes input trivial (just read and broadcast).

[geth_newdate](#) (`./share/module_date_time`) gets the new date based on the old date and the frequency interval, in seconds, between two consecutive input analysis/forecasts. This information is used in the `wrfbdy` file. Subroutine `output_boundary` in `./share/module_io_domain` calls Subroutine `output_wrf` located in `./share/module_io_wrf`. The bulk of the writing is done using this `output_wrf` subroutine. The first part of this routine gets information, including the domain configuration, most of the namelist and namelist derived data. The “`wrf_put`” routines following that writes out those information as global attributes onto the `wrfbdy` file. Finally, the four-dimensional data of a variable at a given boundary is written in Registry generated `wrf_bdyout.inc` using high level package independent WRFIO routine. We explain these details, here, further.

A simple diagnosis of a `wrfbdy` file (perhaps a netcdf format is ideal for this), will illustrate that this file is made up of three parts

(1) global attributes are written using `wrf_put_dom` in [output_wrf](#) (`./share/module_io_wrf.F`). Of specific importance for writing LBC files are calls to `wrf_put_dom_td_char`, i.e.,

```
IF ( switch .EQ. boundary_only ) THEN

CALL ESMF_TimeIntervalSet( bdy_increment, S=NINT(config_flags%bdyfrq),rc=rc)

next_time = current_time + bdy_increment

CALL ESMF_TimeGetString( next_time, next_datestr, rc=rc )

CALL wrf_put_dom_td_char ( fid , 'THISBDYTIME' , current_date(1:19), current_date(1:19), ierr )

CALL wrf_put_dom_td_char ( fid , 'NEXTBDYTIME' , current_date(1:19), next_datestr(1:19), ierr )

ENDIF
```

In the above part of code, based on the `current_date` that is overwritten for each of the analysis loop in the real code, the ESMF time manager and the `bdy_increment`, an array of time starting with “THISBDYTIME” to the end of the simulation and another array starting with the next subsequent analysis time, i.e., “NEXTBDYTIME”, is written out as shown in the example below:

THISBDYTIME	NEXTBDYTIME
"2003-09-15_12:00:00",	"2003-09-15_18:00:00"
"2003-09-15_18:00:00",	"2003-09-16_00:00:00",
"2003-09-16_00:00:00",	"2003-09-16_06:00:00",

"2003-09-16_06:00:00" "2003-09-16_12:00:00",

(2)LBC data: Towards the end of [output_wrf](#), a call is made [wrf_bdyout](#) (./share/wrf_bdyout.F). In this routine, wrf_bdyout.inc calls for a series of Registry generated high-level [wrf_ext_write_field](#). These are the routines that are used to write the LBC data.

(3)Information about the (four dimensional) variables: The following information for a variable, say TBTYS in NMM core, will be found in wrfbdy file:

```
float TBTYS(Time, bdy_width, bottom_top, west_east) ;
```

```
TBTYS:FieldType = 104 ;
```

```
TBTYS:MemoryOrder = "YSZ" ;
```

```
TBTYS:description = "-" ;
```

```
TBTYS:units = "-" ;
```

```
TBTYS:stagger = "" ;
```

These attributes are arguments to the `wrf_ext_write_field` routine that is called, for example, from `share/wrf_bdyout.F`. The meaning of the specific fields

`FieldType` : these integer constants are defined in `external/io_netcdf/wrf_io_flags.h`

`MemoryOrder` : YS means boundary at the start of the Y dimension; Z means has multiple vertical layers

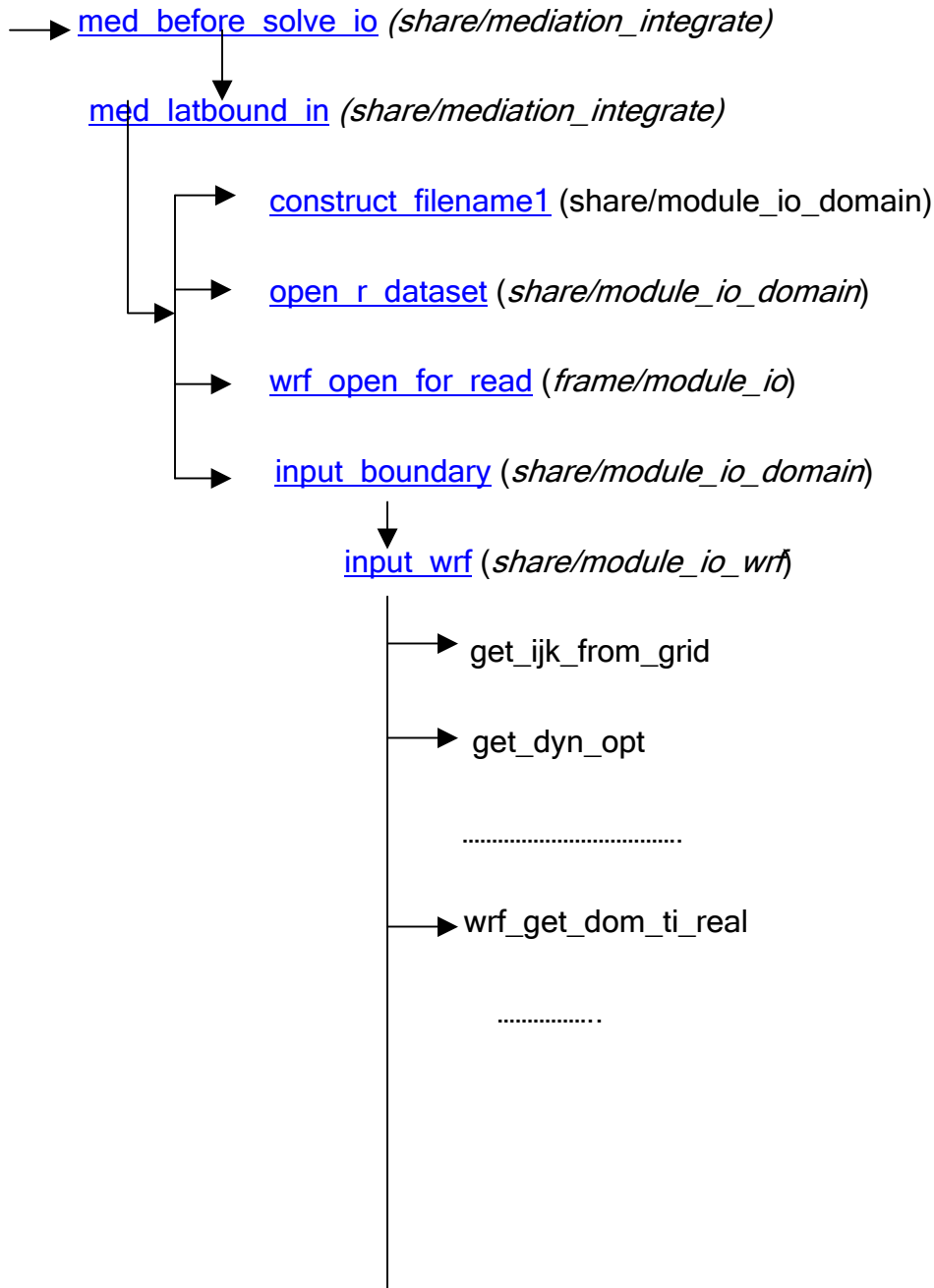
`Stagger ""`: means not staggered in either X, Y, or Z

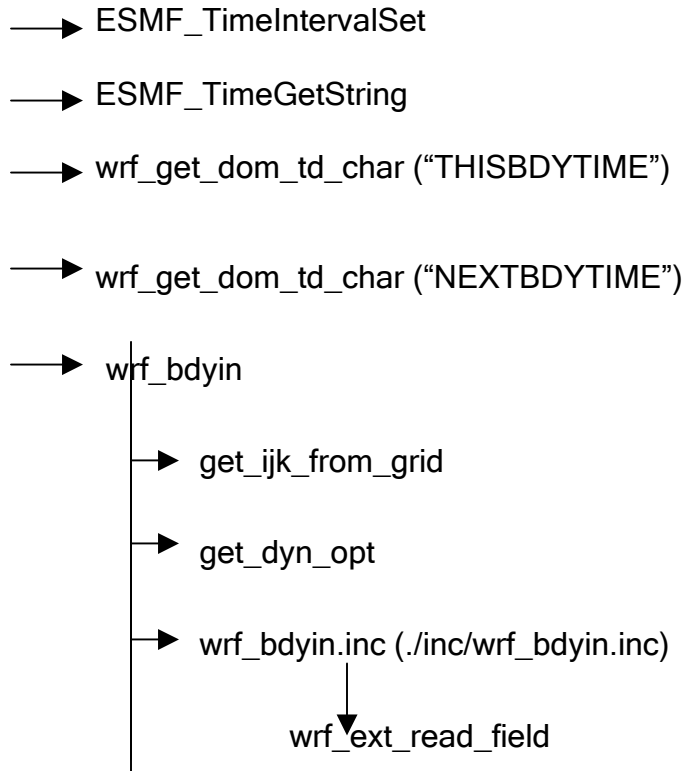
Also, `bdy_width`, `bottom_top`, and `west_east` are the `DimNames` argument passed to `wrf_ext_write_field`

(3) reading of wrfbdy inputs

The aim here is to read the four dimensional data array from the wrfbdy file onto the dynamical framework within the WRF model. The structure of this data array was described in section 2.

recursive subroutine `integrate` (*frame/module_integrate*)





[med before solve io](#) in *share/mediation_integrate* calls for [med latbound in](#) (this is perhaps the last call that is made to io's in [med before solve io](#).) Depending on the type of boundary condition that is set in the namelist.input file, if "specified" as .TRUE. in the namelist.input file, the wrfbdy files are read in at intervals specified in the input file. Note that the wrfbdy files are only read in for the head grid (i.e., grid%id=1). The first part of [med latbound in](#) just checks if the ESMF clock time coincides with LBC reading time. If the time coincides, both the variable at the boundary and their tendency between the fixed interval are read. The steps leading to the reading of this data can be considered as a series of operations (subroutine calls) very similar to the sequence leading to the writing of wrfbdy files explained in section 2. However, note that the "wrf_put" calls in the output_wrf subroutine are simply replaced by "wrf_get" calls in the input_wrf subroutine. Also, wrf_ext_read_field are used within wrf_bdyin.inc.

(4)Using real boundary conditions

Since all state state variables and LBC's are readily available in the solver routine, the four dimensional arrays that are used to store field at the boundary (say, field_bdy) and their time tendency (say, field_tbdy), determined by the frequency of the input data may be used in the model by making appropriate calls in the solver. The following, simple, serial, code, not standard to any of the available core, but tested within an A-grid framework, on a single processor explains how the arrays of a variable and their tendencies may be effectively used in a model:

```

SUBROUTINE update_bc( a, b,                & ! a : output ; b : model variable
                    field_bdy, field_tbdy, & ! LBC array and their tendency
                    spec_bdy_width, spec_zone, & ! pre specified from namelist
                    ijds, ijde,           & ! min/max(id,jd)
                    dtbc,                  & ! time since wrfbdy was read
                    ids,ide, jds,jde, kds,kde, &
                    ims,ime, jms,jme, kms,kme, &
                    lts,ite, jts,jte, kts,kte )

```

```

USE module_state_description

```

```

IMPLICIT NONE

```

```

! i/o

```

```

INTEGER, INTENT(IN) :: spec_bdy_width, spec_zone, ijds,ijde

```

```

INTEGER, INTENT(IN) :: ids,ide, jds,jde, kds,kde, ims,ime, jms,jme, kms,kme

```


ENDDO

ENDDO

ENDDO

! Y-boundary

.....

.....

END SUBROUTINE update_bc

!-----