APPENDIX A: Supported I/O File Formats

There are six types of I/O files used by or created by this package: grid, interface, restart, radiation, BC, and data files. Grid files define the discretized computational geometry of the problem, and are discussed in Section XX. Interface files describe how multiblock grids abut each other in computational space, and are discussed in Section XX. Restart files are saved periodically by the CFD code, and are used (as the name suggests) to restart the problem and/or to post-process the solution. See Section XX for more information. Radiation files are used to enable loose coupling between DPLR and a flowfield radiation analysis tool. See Section XX for more information. BC (boundary condition) files are used to specify various types of pointwise boundary conditions and/or TPS material maps. See Section XX for more information. Finally, data files are generated by POSTFLOW to enable post-processing and data analysis of the solution. See Section XX for more information.

The DPLR package supports several different file I/O formats, as discussed in this section. Not all file formats are compatible with all codes; for example DPLR2D, DPLR3D, and POSTFLOW cannot read plot3d formatted files. Permissible file formats for each code are discussed in the section describing the code. Each file format is assigned a unique number and a unique file suffix that are common across the package. File format numbers are discussed below and summarized in Table A1.

The first digit (if any) of the file format number specifies the data-storage type. A zero for the first digit indicates a file written as machine-specific unformatted files. In general this type of file should be avoided if portability is desired, since an unformatted file created by one machine type cannot in general be read by another. A one for the first digit indicates a file written in XDR format. XDR files are binary, but are written such that they can be read on any machine. This is the recommended storage type for large files, including grid and restart files. In order to read or write XDR files, the *fxdr* libraries must be installed on your computer and linked to DPLR during compilation; see Section XX for more information. A two for the first digit indicates an ASCII file. ASCII files are much larger than binary files, and should be avoided when possible. However, ASCII plot3d files are frequently used for grid input, since they are portable and can be written by most commercial grid generation packages. A three for the first digit indicates a gzipped ASCII file. This format is currently used only for output of plot3d data from POSTFLOW.

The second digit of the file format number indicates the type of file. A one for the second digit indicates a parallel archival I/O file for use with DPLR. This is the preferred file type for grid, restart, radiation, and BC files that are to be read by DPLR. A two for the second digit indicates a plot3d grid or q-file. A three for the second digit indicates a plot3d grid or function file. Plot3d files cannot be read or written by DPLR2D or DPLR3D, but are frequently used to import data from or export data to other programs. A four for the second digit indicates a parallel multi-file grid or restart file; note that this file type is no longer supported by DPLR. A five for the second digit indicates a TECPLOT block file. A six for the second digit indicates a TECPLOT point file. TECPLOT data files are output by POSTFLOW for post-processing purposes, but cannot be read

as input by any of the codes in this package. In order to create binary TECPLOT files, the TECPLOT I/O library must be properly installed and linked to DPLR. See Section XX for more information.

Note that, while they share many common subroutines, DPLR2D and DPLR3D are separate codes, and as such require properly dimensioned input. The most common misconception here is that DPLR2D reads a three-dimensional grid file, with the third dimension set to 1 and all *z*-coordinates set to zero. This is not the case. When preparing a grid for DPLR2D it must be in 2D format. For example, if a plot3D grid is prepared for a 2D simulation, it should be in plot3D two-dimensional format, and not three-dimensional format. If a three-dimensional grid is read as input to FCONVERT with `idim` = 2 the results will be unpredictable, but almost assuredly will not be what the user intends.

Table A1 Allowed file formats

Format	Description	File Type	Suffix
1	Unformatted Parallel	grid	pgrd
		restart	psln
		BC	pbcf
		radiation	prdf
11	XDR Parallel	grid	pgrx
		restart	pslx
		BC	pbcx
		radiation	prdx
21	ASCII Parallel	grid	pgra
		restart	psla
		BC	pbca
		radiation	prda
2	Unformatted Plot3D	grid	gu
		flow	qu
12	XDR Plot3D	grid	gx
		flow	qx
22	ASCII Plot3D	grid	g
		flow	q
32	Gzipped ASCII Plot3D	grid	gz
		flow	qz
3	Unformatted Plot3D	grid	gu

		flow	fu
13	XDR Plot3D	grid flow	gx fx
23	ASCII Plot3D	grid flow	g f
33	Gzipped ASCII Plot3D	grid flow	gz fz
5	Binary Tecplot Block		plt
25	ASCII Tecplot Block		dat
6	Binary Tecplot Point		plt
26	ASCII Tecplot Point		dat

APPENDIX F: More on Zonal Interfaces

There are three possible types of zonal interfaces in 3D grids. The most important type are the face interfaces, which are described in detail in Section XX. However, other types of interfaces, known as edge and corner interfaces, can also be required to fully describe the connectivity of a multi-block grid file. Fortunately FCONVERT is capable of determining edge and corner interfaces for a multi-block grid automatically, and thus this information does not need to be included in the input interface file. However, a discussion of these interfaces is provided here for the interested reader.

Face interfaces define cell face abutment across zonal boundaries and are necessary to permit proper convection and to maintain uniform high-order accuracy of the Euler flux extrapolation across zonal boundaries. Face interfaces of the original grid file must be computed by hand and provided as input to FCONVERT. However, additional face interfaces that result from parallel decomposition can be computed automatically by FCONVERT. As an example, consider Fig. F1 in which a single block with 8 × 8 × 8 computational cells has been decomposed into four by performing a 2 × 2 × 1 decomposition. This decomposition produces four ordinary face interfaces connecting grid blocks 1-2, 1-3, 2-4, and 3-4, as shown in the figure. The face zonal interfaces that would be generated from such a decomposition can be viewed by setting `ouint` = 1 in the input deck. The resulting face interfaces are:

```
----------------------------------------
Zonal Boundary #  1
   nz nface ndr1 nst1 nen1 ndr2 nst2 nen2
    1    2    2    1    4    3    1    8
    2    1    2    1    4    3    1    8

 ----------------------------------------
Zonal Boundary #  2
   nz nface ndr1 nst1 nen1 ndr2 nst2 nen2
    1    4    1    1    4    3    1    8
    3    3    1    1    4    3    1    8

 ----------------------------------------
Zonal Boundary #  3
   nz nface ndr1 nst1 nen1 ndr2 nst2 nen2
    2    4    1    1    4    3    1    8
    4    3    1    1    4    3    1    8

 ----------------------------------------
Zonal Boundary #  4
   nz nface ndr1 nst1 nen1 ndr2 nst2 nen2
    3    2    2    1    4    3    1    8
    4    1    2    1    4    3    1    8
```

Edge interfaces occur in both 2D and 3D grids and are used to specify the connectivity of two grid blocks that abut at a single point (in a 2D grid) or along a single line (in a 3D grid). In the example above, in addition to the four face interfaces a pair of edge interfaces are generated, connecting blocks 1-4 and 2-3. These edge interfaces are indicated by two-way arrows in Fig. F1. Edge interfaces that have been automatically computed by FCONVERT can be viewed if desired by setting `ouint` = 11 in the input deck. The edge interfaces generated in this example look like this:

```
----------------------------------------
Edge Zonal Boundary #  1
   nz nface ndr1 nst1 nen1 ndr2 nst2 nen2
    1     4     1     4     4     3     1     8
    4     3     1     1     1     3     1     8

 ----------------------------------------
Edge Zonal Boundary #  2
   nz nface ndr1 nst1 nen1 ndr2 nst2 nen2
    2     4     1     1     1     3     1     8
    3     3     1     4     4     3     1     8
```

where the range and extent specifiers identify the cells that are adjacent to each other across the zonal edge.

Edge interfaces are not relevant for the computation of the Euler fluxes, since the stencil for computing the Euler fluxes is a cross (See Fig. XX in Section XX). Therefore the Euler fluxes in block #1 can be completely determined without any knowledge of the flow quantities in block #4, and vice versa. However, computation of the full viscous fluxes at a cell face requires information from a full 3 × 3 box stencil (See Fig. XX in Section XX). Therefore, in order to compute the viscous fluxes in block #1, knowledge of the flow values in a single cell in block #4 is required.

If we now consider the case where we split the original grid into eight blocks by performing a 2 × 2 × 2 decomposition (Fig. F2) we generate 12 face and 12 edge interfaces (this can be determined by examination of Fig. F2, or by setting up the test in FCONVERT). In addition, four corner interfaces are generated, connecting blocks 1-8, 2-7, 3-6, and 4-5. Corner interfaces in 3D grids can be considered as analogous to edge interfaces in 2D grids, in that they define only a single computational cell that must be shared between the two grid blocks. Like edge interfaces, corner interfaces are not required for the computation of the Euler fluxes, but are necessary for the computation of the full viscous flux.

The preceding examples of edge and corner interfaces were all created along Cartesian interfaces between two grid blocks. The resulting interfaces are two-way, ie. information transfer is

required in both directions across the specified boundary. It is also possible to generate Cartesian edge and corner interfaces that involve only one-way information transfer. As an example, consider the five-block grid depicted in Fig. F3. This grid has a total of seven face and six edge interfaces. The six edge interfaces that were created by this decomposition are indicated by arrows in the figure. Each of these interfaces is a one-way interface, as indicated by the direction of the arrow. The reason for this can be explained by looking at one of the edge interfaces in more detail. If we look at the edge interface between block #1 and block #4, we can see that block #4 gets all the information required from block #1 from the specified face interfaces. The additional information required to compute the viscous fluxes in block #4 is determined from the face interface shared with block #2. However, block #1 does *not* get all the information required from the face interface with block #4; an additional cell of overlap is required to determine the viscous flux. Therefore a one-way edge interface, that passes information from block #4 to block #1, is required. As discussed in Section XX, one-way interfaces are defined by putting a negative sign in front of the block number of the data receiver. The edge interface between block #1 and block #4 looks like this:

```
 ----------------------------------------
Edge Zonal Boundary #  1
   nz nface ndr1 nst1 nen1 ndr2 nst2 nen2
   -1     4     1     4     4     3     1     4
    4     3     1     5     5     3     1     4
```

One-way edge interfaces are also generated at non-Cartesian boundaries between grids. As an example, consider Fig. F4a. In this case the L-shaped block #2 abuts block #1 along two faces. The face interfaces for this topography are clearly defined, and thus there is no ambiguity in the computation of the Euler fluxes. However, if we closely examine the *imin-jmax* corner in block #1 we notice that it is non-Cartesian; unlike all other interior grid points there are only three (rather than the Cartesian four) grid lines emanating from this point. This is a problem, because the computation of the viscous flux in the *imin-jmax* corner of block #1 requires a Cartesian stencil. From Fig F4a it is apparent that the face interfaces previously defined for this case are not sufficient to fully specify the Cartesian stencil required for the viscous fluxes in block #1; an edge interface is required between block #1 and #2 to supply the necessary information. On the other hand, block #2 *does* have sufficient information from the defined face interfaces. Therefore, what is required is a one-way interface that passes information from block #2 to block #1, but not the other way.

Another interesting problem at non-Cartesian corners arises when we attempt to determine the computational cell in block #2 that should provide data to block #1; the choice is not unique. This can be seen in Fig. F4b, in which we divide the L-shaped block into two. The dashed lines in this figure represent the dummy cells for each block. Only a single row of dummy cells has been shown here, for simplicities sake. The star in block #1 represents the computational cell, which for which flowfield information is required. By a simple mapping exersize it can be determined that there are two possible sources for this information, as indicated by the symbols

in block #2. Which is the "correct" source depends on the method used to determine it. Fortunately, in practice it is not important which of the two possible sources is used, only that a consistent methodology is employed to determine the "correct" source. FCONVERT will automatically determine the correct source for this type of edge interface according to its own internal logic rules. For the case depicted in Fig. F4, FCONVERT will generate the following edge interface:

```
----------------------------------------
Edge Zonal Boundary #  1
   nz nface ndr1 nst1 nen1 ndr2 nst2 nen2
    1     3     1     5     5     3     1     4
   -2     1     2     6     6     3     1     4
```

Other types of non-Cartesian interfaces, with five or more lines emanating from a grid point in 2D are also possible; FCONVERT can handle all of these edge types.

As a final note, while FCONVERT is capable of generating all necessary face, edge, and corner interfaces for any requested parallel decomposition, complex decompositions can result in a large number of interfaces. These interfaces are transparent to the user, but it is important to realize that each interface results in a send-receive message pair when the problem is run in DPLR. Large numbers of MPI messages can adversely affect the computational performance of the method. Therefore it is a good idea to perform parallel decomposition in the simplest manner possible when running FCONVERT.

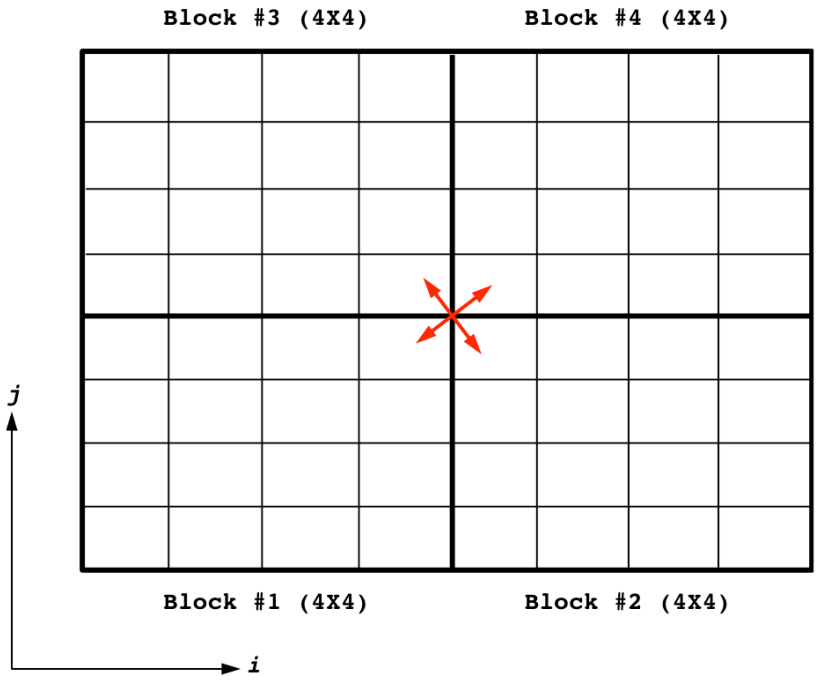

Figure F1 Edge Zonal interface example #1.

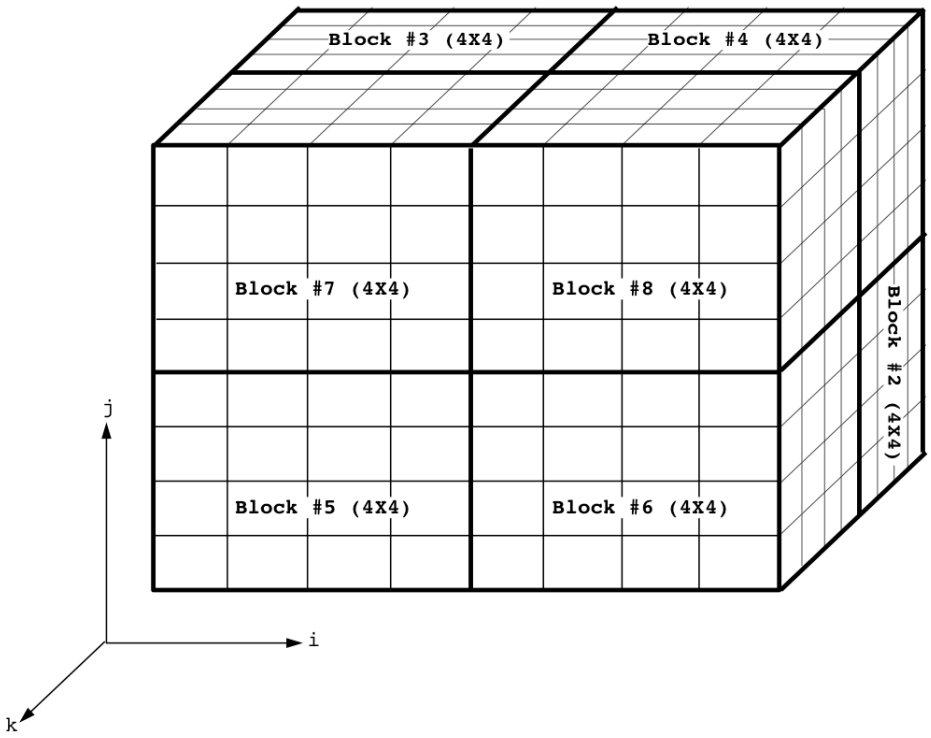

Fig. F2 Edge Zonal interface example #2.

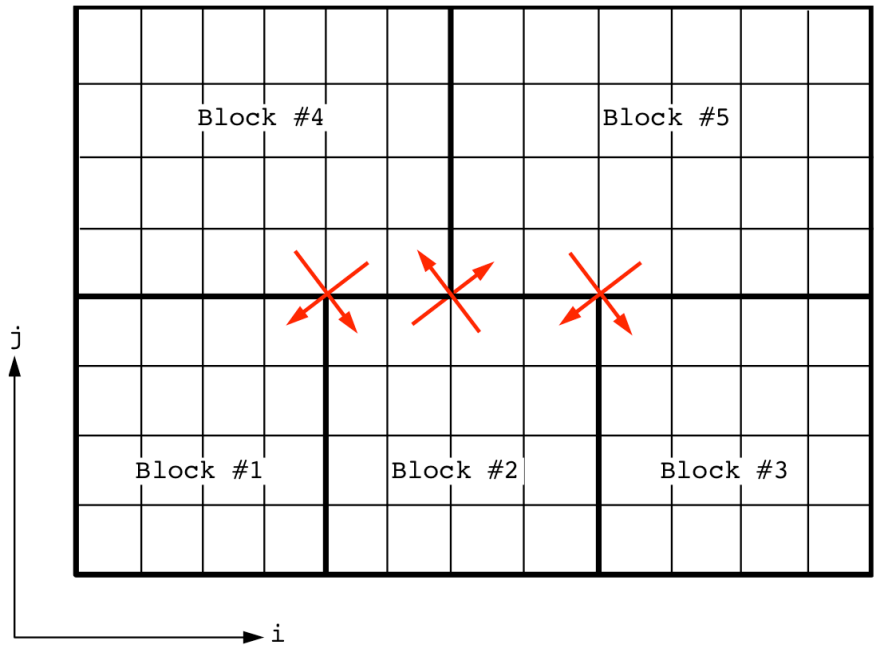

Fig. F3 Edge Zonal interface example #3.

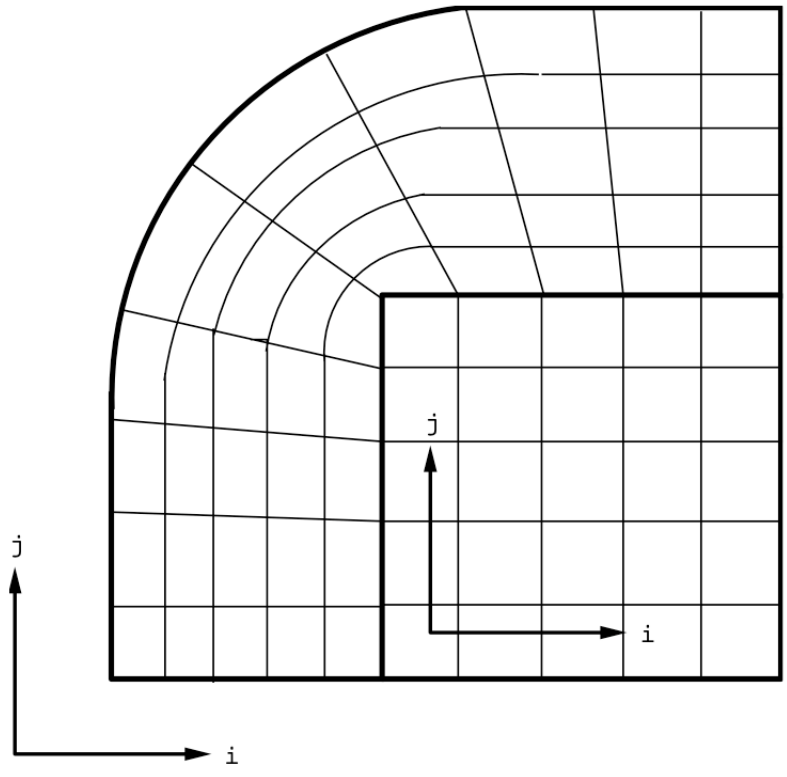

Fig. F4a Edge Zonal interface example #4.

APPENDIX P: More on POSTFLOW Output Variables

This appendix is intended to provide additional information about some of the possible output variables from POSTFLOW. All output variables are expressed either as non-dimensional quantities, or in SI units. DPLR does not support English units.

The output variables in POSTFLOW are selected via the `ivarp` integer array, where each output variable is assigned a unique integer quantity. These integers are a superset of those defined in the *Plot3d* and *GASP* programs. A complete listing of all possible output variables is provided in the Users Manual for POSTFLOW, in this appendix we provide more detailed information about some of these quantities.

Grid-Related Variables

11 path length along grid lines in *i*-direction (si)
12 path length along grid lines in *j*-direction (sj)
13 path length along grid lines in *k*-direction (sk)

Pathlength is determined by computing the distance from grid point to grid point in the mesh along the selected coordinate direction. For example, if `ivarp` = 11 is selected, POSTFLOW will compute the pathlength for each constant *i* line in the output datasets. The pathlength is assumed to begin at zero for *ijk* = 1 and increases for increasing index.

21 *body normal distance (dn)

The body normal distance at a surface is defined as the distance from the cell center of the first interior cell to the face center on the surface. This is the distance used in the first-order approximations of derivatives, as well as that used to define y^+ (`ivarp` = 581), and the cell Reynolds number (`ivarp` = 59).

22 *deviation from orthogonality [deg.] (dev)

This is defined as the number of degrees the surface-normal grid lines deviate from perfect orthogonality. For `interp` = 1 this value represents a local average interpolated to the face center. The primary use of this output variable is as a measure of overall grid quality (orthogonality is desired at all body surfaces, but is generally unimportant at flow-through boundaries).

Mixture Transport Properties

59 cell Reynolds number (Re_c)

The cell Reynolds number is defined as

$$Re_c = \frac{(a+V)\Delta\eta}{\nu}$$

where a is the sound speed, V is the local velocity magnitude, $\Delta\eta$ is the body normal distance (`ivarp` = 21), and ν is the kinematic viscosity. The cell Reynolds number is typically used as a way to judge the adequacy of the near-wall spacing in a boundary layer. $Re_c < 5$ is generally sufficient to ensure accurate heat transfer and skin friction.

Transport Properties

86 laminar Lewis number (Le)
96 turbulent Lewis number (Le_t)

The Lewis number Le is defined as:

$$Le = \rho D C_p / \kappa$$

where ρ is the mixture density, D is the binary diffusion coefficient, Cp is the total specific heat at constant pressure, and κ is the thermal conductivity.

87 laminar Schmidt number (Sc)
97 turbulent Schmidt number (Sc_t)

The Schmidt number Sc is defined as:

$$Sc = \frac{\mu}{\rho D}$$

where μ is the mixture viscosity, ρ is the mixture density, and D is the binary diffusion coefficient.

88 laminar Prandtl number (Pr)
98 turbulent Prandtl number (Pr_t)

The Prandtl number Pr is defined as:

$$Pr = \mu C_p / \kappa$$

where μ is the mixture viscosity, Cp is the total specific heat at constant pressure, and κ is the thermal conductivity.

Mixture Flow Properties

102 stagnation mixture density (r_o)
112 stagnation pressure (p_o)
122 stagnation temperature (T_o)

Stagnation quantities (density, pressure, and temperature) are computed assuming isentropic relations, and thus are not valid for a flowfield with varying isentropic exponent (γ). The stagnation quantities are defined as:

$$\rho_o = \rho S^{\frac{1}{\gamma-1}}$$

$$p_o = p S^{\frac{\gamma}{\gamma-1}}$$

$$T_o = T^S$$

where S is the entropy, defined below.

111 dynamic pressure (Q)

The dynamic pressure Q is simply

$$Q = \rho V^2 / 2$$

114 pressure coefficient (C_p)

The pressure coefficient is defined as

$$(p - p_\infty) / Q_\infty$$

where Q_∞ is the freestream dynamic pressure.

121 bulk temperature (T_b)

The bulk temperature is defined as in AIAA Paper No. 2001-2886:

$$T_b = \frac{V^2}{2C_p}$$

180 degree of ionization (zeta)

$$\zeta = n_e / n_t$$

Surface Properties

512 heat transfer coefficient in mass flux units (Chm)

This is the heat transfer coefficient expressed in kg/m^2•s for use with FIAT.

$$C_{hm} = \frac{q}{\left(h_\infty - h_w\right)}$$

520 radiative equilibrium heat transfer (Qeq)

$$q_{eq} = \varepsilon \sigma T_w^4$$

This is the surface heat transfer as computed using the radiative equilibrium wall formation. In this expression ε is the surface emissivity, σ is the Stefan-Boltzmann constant, and T_w is the surface temperature. This variable is provided mainly as a sanity check to ensure that the computed heat transfer agrees with the radiative equilibrium value when a radiative equilibrium wall is specified.

APPENDIX U: PROVIDED DPLR UTILITIES

Several codes or scripts are provided as utilities to the DPLR package. These tools are summarized in this section. All of these tools are located in the "utilities" directory of the DPLR package.

dpconvert

This tool is a *Perl* script provided to change the format of the DPLR input deck. It is provided primarily to enable a rapid conversion of older DPLR input decks to the current release of the software. The script is run from the command line:

```
dpconvert –i old.inp –o new.inp
```

where "old.inp" is the DPLR input deck that is desired to be converted, and "new.inp" is the output (converted) file. At runtime the script will automatically determine the version of the provided DPLR input deck "old.inp" and convert it to the current version. The user can also specify a desired output version number (other than the current version), with the `–V` option.

seqinput

This tool is a *Perl* script provided to sequence a DPLR input deck easily. Its primary function is to divide the grid sizes of each block in the input deck by a prescribed sequencing factor. The script is run from the command line:

```
seqinter –i old.inp –o new.inp –s I:J:K
```

where "old.inp" is the DPLR input deck that is desired to be sequenced, "new.inp" is the output (sequenced) file, and `I:J:K` are the sequencing factors in the *i-, j-,* and *k*-directions. It is assumed that all blocks are sequenced by the same factors. The script will generate the new DPLR input deck, and rename the input grid and restart files with the suffix "-sIJK". These names can easily be changed by the user if desired.

Moment

This tool is a Fortran code that generates integrated force and moment data from an input set of pointwise surface forces. It is meant to be run on data generated by POSTFLOW using the `ouform` = 11 option, which automatically generates an input deck "Moment.inp" in addition to plot3d files will the appropriate data. Once these data have been generated, Moment is run from the command line:

```
Moment < Moment.inp
```

It should be noted that ***Moment*** was originally written as a standalone tool, and has functionality that is not being used in in this mode. See the man pages on redwood (if available) for more information on the utility of this tool.

zbconvert

This tool is a *Perl* script provided to convert zonal interface files between different formats. The script currently supports conversion between the formats used by DPLR, GASP®, and SAGe. The script is run from the command line:

```
zbconvert –i old.inter –o new.inter [-sage –dplr –gasp] (-g
                          grid.g)
```

The script automatically detects the format of the input interface file and converts it to one of the supported formats specified by the `–sage`, `–dplr`, or `–gasp` flags. If the output format is SAGe, one additional input is required; the user must specify the associated ASCII plot3d grid file using the `–g` flag as shown above. This is because SAGe requires knowledge of the grid size in the input deck, and this information is not available in the interface files for either DPLR or GASP.

The primary uses of `zbconvert` are to transform a GASP formatted interfaced file generated by a commercial grid generator program to DPLR format, and to convert a DPLR interface file to SAGe format in preparation for grid adaption.