Data Processing and Error Analysis System (DPEAS)

User's Guide for DPEAS version 3.012

(Revised December 2011)

Prepared by
Andrew S. Jones, Stanley Q. Kidder, and John M. Forsythe
Cooperative Institute for Research in the Atmosphere (CIRA)
Colorado State University
Fort Collins, CO 80523-1375

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AMSU	Advanced Microwave Sounding Unit
ASCII	American Standard Code for Information Interchange
BJS	Batch Job Server
CD-R	Compact Disk-Recordable
CIRA	Cooperative Institute for Research in the Atmosphere
CPE	Cross-Sensor Processing Environment
CPU	Central Processing Unit
DB	Database
DOS	Disk Operating System
DPE	Data Processing Engine
DPEAS	Data Processing and Error Analysis System
EOS	Earth Observing System
F90	Fortran 90
GIF	Graphics Interchange Format
HDF	Hierarchical Data Format
HPF	High Performance Fortran
HVIOS	HDF-EOS Virtual I/O Subsystem
I/O	Input/Output
McIDAS	Man computer Interactive Data Access System
MB	Megabyte
NESDIS	National Environmental Satellite, Data, and Information Service
NRTRFS	Near Real Time Replication File Subsystem
P2P	Peer-to-Peer
RAMDAS	Regional Atmospheric Model Data Assimilation System
SSM	System State Manager
TBUS	TIROS Bulletin United States
TIFF	Tagged Image File Format
TPW	Total Precipitable Water
UNC	Universal Naming Convention
WWW	World Wide Web

1 INTRODUCTION

The Data Processing and Error Analysis System (DPEAS) is a dynamic, parallel data processing system for the merger and analysis of data from multiple satellite sensors. DPEAS was created to overcome the inherent difficulties of working with large volumes of multiple data formats. Among these difficulties are:

- Data from different satellite sensors come in different formats. Thus, different code must be written for each combination of sensors that is desired.
- Satellite data are voluminous both in total number of bytes and in the number of files that must be processed, including backup and archival.
- The computational burden is not uniform. Parallel processing of the data to avoid processing bottle necks is highly desirable.
- Recoding the system for each new application is far too costly and time consuming.

DPEAS has five main aspects designed to overcome these difficulties:

1. The memory-resident data structure is HDF-EOS (currently HDF-EOS version 2.5). All data are translated on input into the HDF-EOS structure, and then processing continues. On output, a simple subroutine call writes the output data in HDF-EOS format. Other output formats are accomplished with format translators. Therefore, processing code is independent of input or output data format.

2. A large number of utilities are included in DPEAS for the handling of satellite data. Due to the common data structures, most of these routines are generic and can operate on many different satellite data types. This improves the reusability of the advanced satellite processing codes.

3. DPEAS automatically assigns computational tasks to free nodes on a cluster of computers to parallelize the data processing.

4. DPEAS has a number of fault-tolerant features to enable the parallel computing system to reroute data flows dynamically in the event of a hardware failure.

5. DPEAS is run using a scripting language, which is a subset of Fortran 90 (F90). All operations are accomplished through subroutine or function calls. Thus the operational data processing is easy to monitor and change.

The original design and development of DPEAS is detailed in Jones and Vonder Haar (2002), which appears in Appendix A. The remainder of this document describes how to use DPEAS.

2 OVERVIEW OF THIS DOCUMENT

This *DPEAS User's Guide* details how to install and use the current capabilities of DPEAS. A companion document, the *DPEAS Programmer's Guide*, tells Fortran 90 programmers how to add new capabilities to DPEAS and how to modify existing capabilities. DPEAS is a key subcomponent of the Cross-Sensor Processing Environment, which is described in another companion document, the *DPEAS Cross-Sensor Processing Environment (CPE) Guide.* Each CPE process is described more fully by an individualized *CPE Process Document*.

The scope of this document is limited to the DPEAS program. It does not contain information about the CPE or of a particular CPE process.

Section 3 of this document covers the installation and configuration of DPEAS for users. The largest section of this document is Section 4, which covers the use of DPEAS. Section 5 of this document covers the advanced features of DPEAS.

3 INSTALLATION AND CONFIGURATION

This section is intended to facilitate the smooth installation and configuration of DPEAS. Three topics are covered: (1) DPEAS system requirements, (2) installing DPEAS, and (3) configuring DPEAS.

3.1 System Requirements for DPEAS

Before DPEAS can be installed, your computer must first be equipped with the following:

- Windows 7 (or greater)
- Sufficient memory and disk space for the specific application. (This information will be supplied by the developer of the application.)

To run DPEAS in a parallel computing mode, all computers should each have the following:

- Batch Job Server 2.1A (see http://www.camelliasoftware.com)

To modify DPEAS, additional requirements are necessary (see the *DPEAS Programmer's Guide*).

3.2 Installing DPEAS

You may install DPEAS from the DPEAS CD-R or from a shared network drive. The instructions below focus on installation from the DPEAS CD-R, but changes to the procedure for a shared network drive are also noted.

1. If you are installing from the DPEAS CD-R, insert the disk into your computer's CD drive, which we will assume is drive `d:`.
2. Open a DOS command window.
3. In the DOS command window, change the directory to the DPEAS CD-R location (e.g., `d:`), or to the networked drive (e.g., `n:, cd \DPEAS`).
    a. Identify a new directory location target to contain DPEAS. We suggest `c:\DPEAS`. (And we assume below that this is your choice. If your choice differs, adjust the following commands accordingly.)
    b. Verify that the `copy_dpeas.bat` file `source` environment variable definition points to the correct DPEAS source directory location. If necessary, edit the `copy_dpeas.bat` file and redefine the source environment variable value.
    c. Notes:
        i. If DPEAS will be used in a parallel computing mode, please use the Universal Naming Convention (UNC) for the DPEAS directory path (e.g. `\\MYCOMPUTER\myshare\DPEAS`, instead of c:\DPEAS).

ii. Multiple DPEAS versions and installations can co-exist simultaneously in different directories (e.g., `c:\DPEAS2`, `c:\DPEAS3`, etc.).

4. Copy the DPEAS files to your computer by entering one of the following two commands into the DOS command window:

   a. For a full DPEAS installation (including source code), enter
   `copy_dpeas c:\DPEAS full`

   b. For a "lite" DPEAS installation (not including source code), enter
   `copy_dpeas_c:\DPEAS lite`

   After either command has been entered, follow the instructions that appear during the installation procedure.

5. Close the DOS command window and remove the DPEAS CD-R. Installation is complete and no reboot is necessary.

6. Additional install command options can be specified. Please enter `copy_dpeas` with no arguments to read an informational message on the available options.

3.3 Configuring DPEAS for Parallel Mode Operation

DPEAS can automatically assign subtasks (e.g., the processing of a single file of data) to different computers in a network or even to a single computer. This parallelism can both increase processing speed and simplify processing. If you do not plan to use parallel mode, you do not need to configure DPEAS.

To configure DPEAS for parallel mode operation, two steps are necessary:

1. Verify the Batch Job Server (BJS) setup.
2. Configure the resource files.

3.3.1 Batch Job Server (BJS) Setup

To use DPEAS in parallel mode on a cluster of computers, the Batch Job Server service must be running on each computer in the cluster. The BJS service acts as an agent that launches the various parallel jobs and performs the necessary job control and security measures.

Verify that BJS is installed and that you have appropriate BJS user privileges. At a minimum your user account should belong to the following local user groups on each computer for which you intend to run DPEAS in parallel mode:

1. "Batch Users"
2. "Batch Job Dir Users"

3.3.2 Resource File Configuration

Parallel processing is controlled by "resource files," which are located at:
`\DPEAS\setup\configuration\resource\`. They are ASCII text files and may be modified at

any time (even while DPEAS is running). Any modifications to the resource files are made effective upon saving the resource file to disk. Depending upon the current processing load, it may take up to one minute before the changes fully propagate through the system.

A default resource file is created when DPEAS is installed. The resource file is named after the target computer with a `.txt` file extension. DPEAS uses the resource files to determine which computer resource should get the next "job block" of the DPEAS input script.

DPEAS performs the parallelization of jobs automatically. There is no parallel programming by the user. When DPEAS encounters a top-level Fortran "DO loop" in the DPEAS input script, it is parallelized (unless the `DPE_SLAVE` routine has been called, in which case the parallelization is disabled).

To modify a resource file, double-click on the resource file to open it with the default text editor. It should look something like this:

```
&RESOURCE_NML
 RESOURCE%CPU = 1
 RESOURCE%CPU_RATING = 2000
 RESOURCE%MEMORY = 1024
 RESOURCE%AVAILABLE(1) = "M T W Th F S Su 00:00:00.00 24:00:00.00"
 /
```

The file contains a Fortran namelist, `RESOURCE_NML`, which describes the capabilities of the particular DPEAS computer resource. The variables in the list are:

`RESOURCE%CPU` is the maximum number of CPUs that DPEAS may use on that particular computer resource. Zero means that that resource (computer) may not be used. For single CPU computers, set `RESOURCE%CPU = 1`.

`RESOURCE%CPU_RATING` is a relative performance rating (approximately the clock speed in megahertz) that is assigned to that particular computer resource. DPEAS uses a modified round-robin load balancing. If a machine is given a higher rating, it moves to the top of the list. An idle computer with the highest CPU rating is given the next pending job block. DPEAS has the ability to skip resources that do not meet pending job block CPU requirements (more information is available below under the `DPE_REQUIREMENTS` routine description in section 4.1.1.9).

`RESOURCE%MEMORY` specifies the maximum available memory (in megabytes). DPEAS has the ability to skip resources that do not meet pending job block memory requirements (more information is available below under the `DPE_REQUIREMENTS` routine description in section 4.1.1.9).

`RESOURCE%AVAILABLE(n)` specifies the time availability constraints of the computer resource as a data array containing the time availability range. This is specified by a simple time entry for each element of the array. There can be multiple time schedules (e.g., available weekends and non-business hours during the week). The available time array must be specified sequentially (e.g., `n = 1`, `2`, `3`, etc.) if multiple time slots are used. Time is specified in local time.

Every computer should have a resource file for itself. The master computer needs a resource file for each computer in the cluster to which it is allowed to assign jobs.

When testing new DPEAS input scripts, it is recommended that you disable parallelization with a call to the `DPE_SLAVE` routine so that you can more easily view the system results. Also, since there is no guarantee that a DPEAS job block will run on the same computer resource, use UNC file path names throughout the DPEAS input script files (e.g., use file names such as: `\\mycomputername\mysharename\mydirectory\myfile.f90`.).

Security is handled at the network domain level, the resource files are used to inform DPEAS of potential resources that are available; they do not grant resources.

4 USING DPEAS

DPEAS can be run from the command line, from within MS Visual Studio, or from a batch file.

- From the command line, run `DPEAS.exe` with an explicit path to this executable's location. `DPEAS.exe` takes a single argument, which is a DPEAS script file that contains instructions to DPEAS. For example, the command

  ```
  \DPEAS\bin\DPEAS_Win32_Release.exe ..\data\input\test.f90
  ```

  will run the release version of DPEAS using the test.f90 script located in the \DPEAS\data\input directory. Section 4.1 describes the DPEAS script files.

- From within MS Visual Studio the Executable and Program Arguments fields under Project Settings are set to the locations of the executable file and input script. DPEAS can be run without recompilation. The input script does not need to be compiled.

- DPEAS can be run from a DOS Batch File quite easily. It is also possible to set program exit return codes using BJS utility commands. This enables DPEAS to change the BJS status for easier error handling behaviors.

  DPEAS must be called with a UNC path naming convention to enable parallel mode execution. Relative path names for the DPEAS input script file are allowed, and are relative to the DPEAS executable directory location.

A major portion of the DPEAS script file tells DPEAS which files to process. Section 4.1 describes the basic DPEAS input script syntax. Section 4.2 describes the data directory structure for DPEAS.

DPEAS is capable of running on several computers (nodes) simultaneously. Section 4.3 describes the DPEAS parallel processing capabilities. To turn off the automatic parallelization of DPEAS input (advisable when first using DPEAS), add `call DPE_SLAVE` at the start of the execution part of the input file.

The output of DPEAS (in addition to the processed data files) is a log file, which tells what DPEAS did and what problems it found while processing the data. Section 4.4 describes the interpretation of log files.

4.1 Script File Description

Script files tell DPEAS what to do. A DPEAS script file is text file in free source form Fortran 90. The script files are given an .F90 extension, but they are not compiled. Rather, DPEAS acts as a Fortran 90 interpreter. Let's consider the following example DPEAS script file.

```
! DPEAS input syntax example
```

```
character (len = 255), pointer :: file (:)            ! file names
character (len = 255) :: input_files = '\\DORADO\E\TEMP\*.dat'
character (len = 255) :: output_dir = '\\ULYSSES\E\HDFEOS\SSMI\'

integer :: i                                          ! dummy integer
integer :: n                                          ! number of files

call get_files (input_files, file, n)
do i = 1, n
   call ssmi2hdfeos (file(i), output_dir)
enddo
```

The first line of the example is simply a comment—and comments should be used liberally to document the script. The 2nd through 5th lines (not counting blank lines) define some variables to be used. (all variables in DPEAS scripts must be defined. `IMPLICIT NONE` is assumed.) The `get_files` routine is one of the DPEAS intrinsic routines, which are described below. `get_files` searches for files that match the string `input_files` and returns a list of these files in the array `file`. The number of files found is returned in variable `n`. The DO loop processes each of the `n` files with the routine `ssmi2hdfeos`. The processed files are placed in the directory specified by the variable `output_dir`.

The DPEAS F90 interpreter implements only a subset of Fortran 90. The complete list of allowed statements is detailed in
`http://lamar.colostate.edu/~asjones/DPEAS/DPEAS_syntax.htm`.

Writing a subroutine cannot be done in the DPEAS script. However, the DPEAS intrinsic routines (see section 4.1.1) cover a wide variety of ways to process satellite data. In addition, other routines, called application-specific routines, can be written, compiled, and integrated into DPEAS to do special tasks (see the *DPEAS Programmer's Guide*). Some application-specific routines are supplied with DPEAS (see section 4.1.2).

Other example input script files are contained under the `C:\DPEAS\examples` directory.

4.1.1 Intrinsic Routines

DPEAS intrinsic routines are the subroutines that constitute the core capabilities of DPEAS. The intrinsic routines described in sections 4.1.1.1 through 4.1.1.29 are organized alphabetically. At the end of each section, a piece of code shows the subroutine interface. To use the routine, set up the arguments, and then call the routine. The description of the routines here is a high-level description so that DPEAS users can understand the script files. To write or significantly modify the script files, consult the *DPEAS Programmer's Guide.*

4.1.1.1 Archive

Purpose: This routine archives (moves) files from a source directory to a target directory. The number of files to be moved is determined by the target file size. This permits archival to size-limited, off-line archival storage. The archive operation is performed on all subdirectories simultaneously in sequential filename order. Additionally, the operation

is capable of handling mirror set replication behaviors. If used, the limit specified is a soft-limit since all files that share the same YYYYDDD (or 7 character) filename prefix are moved in a contiguous manner.

```
subroutine archive (source, target, limit)

character (len = 255), intent (IN) :: source         ! source directory
character (len = 255), intent (IN) :: target         ! target directory
double precision, intent (IN), optional :: limit     ! target size limits (bytes)
```

4.1.1.2 Composite

Purpose: This routine composites an input file according to a specific method and precalculated weights. Methods supported include: 1) "UNIFORM" – uniform weights are used; this results in a simple average of the input data files, and 2) "OVERLAY" – uniform weights are used, then a simple time filter is applied to use only the most recent data in the overlay. In the "OVERLAY" method, the temporal weighting can be controlled by the optional `half_life` and `zero_offset` arguments. The temporal weight is computed as: `exp (-half_life * delta time) + zero_offset`. Negative temporal weights are truncated to zero. By default, or if the `half_life` is defined as zero, a temporal "if statement" is used (i.e., only the most recent data is retained). The `weight_file` argument specifies a HDF file containing static weights to use within the compositing process. If the `weight_file` argument is omitted, uniform static weights are generated dynamically by this routine. If the start time is given, the time prefix of the static output file name may be updated. The start argument is required by the "OVERLAY" method. The start time can be specified as "NOW" if the current time is desired. If the control argument is omitted, the default behavior is to create all possible composite output variables. The control argument is a binary flag that corresponds to the integers specified for each appropriate composite variable (for the specified composite method) in the `dpe_composite_method_module`. The bits are specified with the rightmost bit being the least significant. The output data file is reinitialized with each invocation of this routine. Therefore, prior composite results are not intermixed with the new composite output results.

```
subroutine composite (input_file, output_file, method, weight_file, start, control)

character (len = 255), intent (IN) :: input_file            ! input file name
character (len = 255), intent (INOUT) :: output_file        ! output file name
character (len = 255), intent (IN) :: method                ! composite method
character (len = 255), intent (IN), optional :: weight_file ! weight file name
character (len = 255), intent (IN), optional :: start       ! composite start time
double precision, intent (IN), optional :: half_life        ! temporal half life
double precision, intent (IN), optional :: zero_offset      ! temporal zero offset
integer, intent (IN), optional :: control                   ! output control flag
```

4.1.1.3 Composite_Weights

Purpose: This routine generates the static composite weights for a particular input file. This routine supports the method, "UNIFORM" – uniform weights are used; this generates

a weight file containing weight values of "1" for each valid data element contained in the input data file.

```
subroutine composite_weights (input_file, output_file, method)

character (len = 255), intent (IN) :: input_file            ! input file name
character (len = 255), intent (INOUT) :: output_file        ! output file name
character (len = 255), intent (IN) :: method                ! composite method
```

4.1.1.4 Copyfile

Purpose: This routine copies an existing file to a new file using a modified file name specification. If the file already exists, the copy operation is still performed.

```
subroutine copyfile (input_file, context, file_spec, dir_spec, prefix_spec, &
       suffix_spec, file_type_spec)

character (len = 255), intent (IN) :: input_file          ! input file name
character (len = 255), intent (IN), optional :: context   ! search context
character (len = 255), intent (IN), optional :: file_spec ! output file name
character (len = 255), intent (IN), optional :: dir_spec  ! directory name
character (len = 255), intent (IN), optional :: prefix_spec    ! prefix name
character (len = 255), intent (IN), optional :: suffix_spec    ! suffix name
character (len = 255), intent (IN), optional :: file_type_spec ! file type name
```

4.1.1.5 Deallocate_Hdfeos

Purpose: This routine deallocates an HDF-EOS data file from the DPEAS data structure or, optionally, deletes a component of an HDF-EOS data file such as a grid, a point, or a swath, or a specific data item.

```
subroutine deallocate_hdfeos (input_file, name, grid, point, swath, item)

character (len = 255), intent (IN) :: input_file      ! input file name
character (len = 255), intent (IN), optional :: name  ! grid, point, or swath name
character (len = 255), intent (IN), optional :: grid  ! grid name
character (len = 255), intent (IN), optional :: point ! point name
character (len = 255), intent (IN), optional :: swath ! swath name
character (len = 255), intent (IN), optional :: item  ! data item name
```

4.1.1.6 DPE_Code_Check

Purpose: This routine performs fortran source code style verification. It implements a multipass scan and reports any style discrepancies as error messages. No modification of the input file is performed. Two special code directives can be placed into the fortran source files:

1) `!DPEAS$BASIC_CODE_CHECK`: this only performs the most basic code checking on the rest of the file, normally it is placed at the top of a particularly complex fortran source file, or a file which is not intended for distribution.

2) `!DPEAS$NOSYSTEM_CODE_CHECK`: this skips the system library usage checks, but performs the rest of the code checking on the rest of the file.

3) `!DPEAS$TOGGLE_CODE_CHECK`: this allows toggling of the code checking behaviors and is useful to particularly long/boring sections of codes which might otherwise fail the number of commentless lines scan.

```
subroutine dpe_code_check (input_file)

character (len = 255), intent (IN) :: input_file      ! input file name
```

4.1.1.7 DPE_Mirror

Purpose: This routine performs a full update of a particular mirror set source. To update all available mirror set sources, set `mirror_source` to "`*`". The mirror sets are defined in the DPEAS resource files (see 5.2 for more information). The mirror set information format is:

```
Mirror%source(1) = "\\node name\share name\source directory name\"

Mirror%target(1) = "\\node name\share name\target directory name\"

Mirror%replicate(1) = .TRUE.
```

where, if `mirror%replicate` is `.TRUE.`, then replication is performed (i.e., if target files do not exist on the source, then the target files are deleted). Otherwise, if `mirror%replicate` is `.FALSE.`, the synchronization is performed (i.e., if target files do not exist on the source, then the target files are copied to the source location). The synchronization mode does not delete any files, and is the default setting if `mirror%replicate` is not explicitly defined in the resource file. Use the replication-mirroring mode with caution, since it enables an automatic deletion process.

```
subroutine dpe_mirror (mirror_source)

character (len = 255), intent (IN) :: mirror_source         ! mirror set source
```

4.1.1.8 DPE_Mode_Asynchronous

Purpose: This routine sets the job behavior to be asynchronous at the end of each DO loop (i.e., the next line of code does not wait for the prior active DO loop to complete).

```
subroutine dpe_mode_asynchronous
```

4.1.1.9 DPE_Mode_Sychronous

Purpose: This routine sets the job behavior to synchronize at the end of each DO loop. This is the default behavior in DPEAS.

```
subroutine dpe_mode_synchronous
```

4.1.1.10 DPE_Requirements

Purpose: This routine updates the job resource requirements from their nominal default values. This routine is used to control the DPEAS job submission logic so that only nodes with sufficient resources are sent jobs that require a large amount of resources. Normally, the default resource requirement values are adequate since most jobs are CPU limited. However, sometimes large memory or multi-threaded jobs need additional control. This routine provides that control.

```
subroutine dpe_requirements (cpu, memory)

integer, intent (IN) :: cpu                          ! cpu(s) required
integer, intent (IN) :: memory                       ! memory (MB) required
```

4.1.1.11 DPE_Slave

Purpose: This routine makes the current node a slave node. It is primarily for internal use by the DPEAS system. If used on a master node, it will disable the parallelization of subsequent DO loop blocks.

```
subroutine dpe_slave
```

4.1.1.12 DPE_Test_Call

Purpose: This routine is a test subroutine for debugging the DPEAS argument functions. This is also a good example of a DPEAS subroutine driver.

```
subroutine dpe_test_call (test1, test2, string1, string2)

integer, intent (OUT) :: test1 (3)
integer, intent (INOUT) :: test2
character (len = 30), optional, intent (INOUT) :: string1
character (len = 30), optional, intent (INOUT) :: string2 (2)
```

4.1.1.13 DPE_Test_Hdfeos

Purpose: This routine performs a system test of the HDF-EOS library. A test file named `DPE_TEST_HDFEOS.hdf` is generated in the computer's TEMP directory. If IMAX is specified to be larger than 1, then the array sizes are proportionally enlarged to conduct test using the larger file size.

```
subroutine dpe_test_hdfeos (imax)

integer, intent (IN) :: imax           ! size of test (nominal test is 1)
```

4.1.1.14 DPE_Test_Load

Purpose: This routine performs a system test with the specified CPU and I/O load factors.

```
subroutine dpe_test_load (cpu_io_ratio, load_factor)
```

```
real, intent (IN) :: cpu_io_ratio       ! CPU/IO ratio, range (0.0 – 1.0)
real, intent (IN) :: load_factor        ! load amount, range (0.0 – 1.0)
```

4.1.1.15 DPE_Write_Data_Structure

Purpose: This routine writes the entire contents of the DPEAS data structure list to the default output device as a formatted list. It is useful for debugging the DPEAS data structure.

```
subroutine dpe_write_data_structure
```

4.1.1.16 DPE_Write_DB_Statistics

Purpose: This routine writes the DPEAS statistical summary information from all specified input database directories. It is useful for consolidating multiple DPEAS installation reports and takes up to 99 database directory path names. The directory path should be specified using UNC, e.g., `\\DORADO\D\Jones\Projects\DPEAS\`. Note that if subtotals = `.TRUE.`, then the routine will print all of the database subtotal statistics for each database file it encounters; otherwise, if subtotals = `.FALSE.`, it will not print the subtotal statistics and will merely list the aggregate statistics totals.

```
subroutine dpe_write_db_statistics (subtotals, path01, path02, pathnn…)

logical, intent (IN) :: subtotals              ! logical subtotal flag
character (len = 255), intent (IN) :: path01   ! database directory path 1
character (len = 255), intent (IN) :: path02   ! database directory path 2
character (len = 255), intent (IN) :: path03   ! database directory path 3
```

4.1.1.17 DPE_Write_Mirror_Sets

Purpose: This routine writes all DPEAS mirror set definitions to the default output device as a formatted list. It is useful for debugging the DPEAS mirror set specifications.

```
subroutine dpe_write_mirror_sets
```

4.1.1.18 DPE_Write_Variables

Purpose: This routine writes all internal DPEAS variables to the default output device as a formatted list. It is useful for debugging the DPEAS internal variables.

```
subroutine dpe_write_variables
```

4.1.1.19 Filename

Purpose: This routine constructs a file name specification for use in other routines. It is necessary because the concatenation operator (`//`) is not implemented in the DPEAS script language. To piece together the path, filename, and file extension, `filename` is necessary.

The filename routine starts with the default file specification, `file_spec`, then overrides the various file name segments with the optional filename routine arguments. For example, the directory name argument, `dir_spec`, will replace the directory name portion of the input `file_spec` variable, while the file type name argument `file_type_spec`, replaces the file type (e.g., `.TXT`) of the input `file_spec` variable. The `prefix_spec` and `suffix_spec` can only append new information to the file name segment (e.g., what is left after the directory and file types are removed). This allows file names to be easily generated from existing file names. A common use of `suffix_spec` is to append the `_REMAP` suffix to a file name to visually indicate that it has been remapped via a DPEAS remapping routine (e.g., `\directory_path\filename_REMAP.HDF`). This capability gives the DPEAS user easy control over the naming of output files.

```
subroutine filename (file_name, file_spec, dir_spec, prefix_spec, suffix_spec, &
       file_type_spec)

character (len = 255), intent (OUT) :: file_name                  ! output file name
character (len = 255), intent (IN) :: file_spec                   ! input file name
character (len = 255), intent (IN), optional :: dir_spec          ! directory name
character (len = 255), intent (IN), optional :: prefix_spec       ! prefix name
character (len = 255), intent (IN), optional :: suffix_spec       ! suffix name
character (len = 255), intent (IN), optional :: file_type_spec    ! file type name
```

4.1.1.20 Get_Files

Purpose: This routine returns a list of files that match a target file specification. Note that this routine dynamically allocates space for the files using a pointer data type. A unique search context identifier can optionally be specified so that previously returned file names are not returned in subsequent searches. This is particularly useful for near real-time processing of data streams where the input directory is being constantly updated with new files. Context information is maintained between multiple instances of the program and coordination of the context information between nodes occurs automatically. Thus, all context identification specifications should be unique to each input script.

The `context` argument is used to allow DPEAS to maintain a coherent system processing context information (a history of what has or has not been already processed). The `context` argument is the mechanism DPEAS uses to recall what files have been returned in previous `get_files` invocations. A unique character string identifier (e.g., `PROCESSED_FILES`) can be used which will allow the `get_files` routine to only return "new" files not found or processed previously. The previous accumulated history of `get_files` results are saved under the `\DPEAS\data\work\<context>` directory as simple ASCII placeholder marker files. To flush the context (and perform a full fresh search), either delete the `\DPEAS\data\work\<context>` directory; or to temporarily flush the context, simply call `get_files` without the context argument. Be careful not to inadvertently use the same search context argument more than once, otherwise the context information may be overwritten unintentionally, resulting in loss of the context information. The search

context information in the `\DPEAS\data\work\<context>` directory does not require any user intervention. The information is automatically managed and maintained by DPEAS.

Optionally, the results can be filtered using additional time filter criteria. All time filter criteria are optional and can be specified in many possible combinations. All times are based on the file naming conventions. The time filter should not be applied to non-YYYYDDDHHMMSS... filenames.

Time filter argument definitions:

- Start: The start time of the period can be specified as an ASCII text string in an absolute ('YYYY-DDD HH:MM:SS.CC' or 'YYYY-MON-DD HH:MM:SS.CC' or 'YYYY-MM-DD HH:MM:SS.CC') or delta ('DDD HH:MM:SS.CC') time formats, or by ASCII text file names following the YYYYDDDHHMMSS... nomenclature.
- Period: The period is the length of time per cycle. It is specified by an ASCII text string in delta time format.
- Duration: The duration is the length of the time filter "window". It is specified by an ASCII text string in delta time format.
- Offset: The offset is applied to the start time of the period, and offsets all time window cycles. It is specified by an ASCII text string in delta time format. It can be negative (e.g., "- 01:00:00.0").
- Occurrences: The number of occurrences controls how many cycles are filtered. If set to 0, the file nearest to the center of the time filter "window" is returned from each filter cycle.

For example, suppose we want to get the files that lie between 08:00 and 13:00 hours on day 264 of 2003. The call to `get_files` would be:

```
call get_files(target_file_spec, file, n, START='2003-264 08:00:00.00',
DURATION='0 05:00:00.00')
```

To get the files that lie between 08:00 and 13:00 hours for each day over a ten day period beginning on day 264 of 2003, the call to `get_files` would be:

```
call get_files(search, file, n, START='2003-264 08:00:00.00', PERIOD='1
00:00:00.00', DURATION='0 05:00:00.00', OCCURRENCES=10)
```

If the Period option is used, the Occurrences option should be specified as well.

Time filter argument default values:

- Start: The current time is used.
- Period: An infinite period is used.

- Duration: An infinite duration is used.
- Offset: Set to zero.
- Occurrences: An infinite number of occurrences are tested.

If all time filter arguments are omitted, no time filtering is performed on the files.

Optionally, the file name list can be prefilterd so that file name duplicates are removed. The argument `unique_filename_length` specifies the maximum offset position for the uniqueness test.

Optionally, the nearest matching file can be returned according to the file name nomenclature. The argument, match, specifies the target file name that is to be matched. You may modify the match by using the optional time filter arguments. The match argument supersedes the start filter argument. By default, the `duration` and `offset` arguments are set to search all available files for the match.

```
subroutine get_files (target_file_spec, file, number_of_files)

character (len = 255), intent (IN) :: target_file_spec      ! target file
character (len = 255), pointer :: file (:)                  ! file names
integer, intent (OUT) :: number_of_files                    ! number of files found
character (len = 255), intent (IN), optional :: context     ! context ID
character (len = 255), intent (IN), optional :: start       ! filter start time
character (len = 255), intent (IN), optional :: period      ! filter period
character (len = 255), intent (IN), optional :: duration    ! sample duration
character (len = 255), intent (IN), optional :: offset      ! filter time offset
integer, intent (IN), optional :: occurrences               ! number of periods
integer, intent (IN), optional :: unique_filename_length    ! unique filename length
character (len = 255), intent (IN), optional :: match       ! target to match
```

Note that if `get_files` is called more than once in an input script using the same variable for the `file` argument, then this variable must be deallocated before the next call to get_files where it is used. For example:

```
call get_files (target_file1, file, n) ! uses 'file' as an argument
deallocate (file) ! must deallocate 'file' before the next get_files call that uses it
call get_files (target_file2, file, n) ! uses 'file' again
```

4.1.1.21 Read_Hdfeos

Purpose: This routine reads an HDF-EOS data file and inserts the file into the main DPEAS data structure. The optional `only` argument allows for the selective reading of HDF-EOS data files to minimize unnecessary I/O. The `only` argument is specified as a comma-delimited list of HDF-EOS data field names that are to be read, for example: only = 'Lat,Lon,Chan1'.

```
subroutine read_hdfeos (input_file, only)

character (len = 255), intent (IN) :: input_file            ! input file name
character (len = 255), intent (IN), optional :: only        ! data to read
```

4.1.1.22 Remap

Purpose: This routine remaps an HDF-EOS data file to a specified projection space (given by HDF-EOS projection parameters contained in the `PROJ_FILE` input file). All data contained in the HDF-EOS file are remapped. One-dimensional arrays that have a defined geographical mapping are converted to two-dimensional arrays to reflect the spatial remapping. Note: that this routine promotes the output data item numbertype to either FLOAT32 or FLOAT64 depending on the input data item numbertype. The remapped data are written to an internal DPEAS data structure with the name `output_file`. Data structure names within the HDF-EOS data file are unchanged. This routine performs no explicit I/O. To save the remapped data to a file on the drive use the `WRITE_HDFEOS` routine with the value contained in `output_file` as its argument. If the `bin` argument is `.TRUE.`, use the binning method instead of the interpolation method. The arrays that are (or are not) processed are controlled by the `only2d` argument value.

The following input data arrays are not defined by this routine for remapping under the following `only2d` conditions:

if `only2d = .TRUE.`

1. Arrays with the suffix `_Calibration`.
2. Arrays with the suffix `_Attenna_Pattern_Correction`.
3. Arrays that have non-geographical dimensions.

if `only2d = .FALSE.`

1. No arrays are skipped. Arrays of all ranks are defined. Caution: some multidimensional arrays with non-geographical dimensions can become quite large.

```
subroutine remap (input_file, proj_file, output_file, bin, only2d)

character (len = 255), intent (IN) :: input_file           ! input file name
character (len = 255), intent (IN) :: proj_file            ! projection file name
character (len = 255), intent (INOUT) :: output_file       ! output file name
logical, intent (IN), optional :: bin                      ! bin control flag
logical, intent (IN), optional :: only2d                   ! only2D control flag
```

4.1.1.23 System

Purpose: This routine passes a command to the operating system and then pauses until the command is done.

```
subroutine system (command)

character (len = MAX_SYSTEM_COMMAND), intent (IN) :: command      ! command line
```

4.1.1.24 Time

Purpose: This routine writes a message containing the current local time and the elapsed time from the previous call.

```
subroutine time
```

4.1.1.25 Write_GIF

Purpose: This routine writes a DPEAS data structure data item to a digital image in GIF 87a format. Optionally, this routine can scale the data and fill in missing values in the image with the fillcount value. This routine will not create an output image file if the data field contains only missing data flags.

```
subroutine write_gif (input_file, name, item, output_dir, min, max, inc, fillcount)

character (len = 255), intent (IN) :: input_file     ! input file name
character (len = 255), intent (IN) :: name           ! swath or grid name
character (len = 255), intent (IN) :: item           ! data item name
character (len = 255), intent (IN) :: output_dir     ! output directory
real, intent (IN), optional :: min                   ! minimum value
real, intent (IN), optional :: max                   ! maximum value
real, intent (IN), optional :: inc                   ! increment value
integer, intent (IN), optional :: fillcount          ! fill count value
```

4.1.1.26 Write_Hdfeos

Purpose: This routine writes an HDF-EOS data file from the main DPEAS data structure to disk. All data components of the HDF-EOS data file are written.

```
subroutine write_hdfeos (output_file)

character (len = 255), intent (IN) :: output_file    ! output file name
```

4.1.1.27 Write_Ramdas

Purpose: This routine writes a DPEAS data structure data item to a tab-delimited text file in the RAMDAS output format.

```
subroutine write_ramdas (input_file, name, item, output_dir)

character (len = 255), intent (IN) :: input_file     ! input file name
character (len = 255), intent (IN) :: name           ! swath or grid name
character (len = 255), intent (IN) :: item           ! data item name
character (len = 255), intent (IN) :: output_dir     ! output directory
```

4.1.1.28 Write_Text

Purpose: This routine writes a DPEAS data structure data item to a text file in tab-delimited ASCII format. The first column of data is that which is specified by the data item

name. All associated data items are written in the following columns. Currently, only arrays that match the target data item array in rank and dimension size are written.

```
subroutine write_text (input_file, name, item, output_dir)

character (len = 255), intent (IN) :: input_file     ! input file name
character (len = 255), intent (IN) :: name           ! swath or grid name
character (len = 255), intent (IN) :: item           ! data item name
character (len = 255), intent (IN) :: output_dir     ! output directory
```

4.1.1.29 Write_TIFF

Purpose: This routine writes a DPEAS data structure data item to a digital image in TIFF format. Optionally, this routine can scale the data and fill in missing values in the image with the fillcount value. This routine will not create an output image file if the data field contains only missing data flags.

```
subroutine write_tiff (input_file, name, item, output_dir, min, max, inc, fillcount)

character (len = 255), intent (IN) :: input_file     ! input file name
character (len = 255), intent (IN) :: name           ! swath or grid name
character (len = 255), intent (IN) :: item           ! data item name
character (len = 255), intent (IN) :: output_dir     ! output directory
real, intent (IN), optional :: min                   ! minimum value
real, intent (IN), optional :: max                   ! maximum value
real, intent (IN), optional :: inc                   ! increment value
integer, intent (IN), optional :: fillcount          ! fill count value
```

4.1.1.30 WWW_Update

Purpose: This routine purges any old files and updates the web directory information. This routine uses the file's last-written date for purging purposes. If the keep argument is not present, no files are purged. The directory name specification can include a filename, but only the directory name information is used. If the plot argument is `.TRUE.`, a simple plot is made of the file times (as determined by the filename nomenclature) to the standard output, and some diagnostics are printed for data gaps and overlaps. The plot capability is especially useful to quickly examine data sampling characteristics. No plots are generated by default (i.e., `plot = .FALSE.`).

```
subroutine www_update (www_dir, dir_spec, keep)

character (len = 255), intent (IN) :: www_dir        ! WWW output directory
character (len = 255), intent (IN) :: dir_spec       ! directory name specification
real, intent (IN), optional :: keep                  ! number of days to keep
logical, intent (IN), optional :: plot               ! plot the file times
```

4.1.2 Format Translator Routines

4.1.2.1 AGRMET2HDFEOS

Purpose: Convert AFWA AGRMET model output from GRIB format to an HDF-EOS grid format. This routine can handle 3HR and polar stereographic GRIB files at "8th mesh". This routine automatically generates a file name based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure. The DPEAS_SAT_NAME and DPEAS_SENSOR_NAME are used to contain the model name and the model output type information.

```
subroutine agrmet2hdfeos (input_file, output_dir, output_file)

character (len = MAX_PATH), intent (IN) :: input_file             ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir             ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
```

4.1.2.2 AMSU2HDFEOS

Purpose: Convert an AMSU data file in NESDIS HDFEOS format to a "pure" HDF-EOS format. This routine can handle AMSU-A and AMSU-B sensor data files. This routine automatically generates a file name based on the data time. All available multispectral data matching the input file specification is combined into one multichannel HDF-EOS file. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure. If called without the sat_number argument, a default value of "15" is used.

```
subroutine amsu2hdfeos (input_file, output_dir, output_file)

character (len = MAX_PATH), intent (IN) :: input_file             ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir             ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
```

4.1.2.3 AVHRR2HDFEOS

Purpose: Convert an AVHRR GAC or LAC data file in McIDAS format to HDF-EOS. This routine automatically generates a file name based on the data time. All available multispectral data matching the input file specification is combined into one multichannel HDF-EOS file. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created

HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine avhrr2hdfeos (input_file, output_dir, output_file, min_lat, max_lat,
min_lon, max_lon, min_time, max_time)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
real, intent (IN), optional :: min_lat                             ! minimum latitude
real, intent (IN), optional :: max_lat                             ! maximum latitude
real, intent (IN), optional :: min_lon                             ! minimum longitude
real, intent (IN), optional :: max_lon                             ! maximum longitude
character (len = 23), intent (IN), optional :: min_time            ! minimum time
character (len = 23), intent (IN), optional :: max_time            ! maximum time
```

4.1.2.4 GDAS2HDFEOS

Purpose: Convert a GDAS data file in GRIB format to HDF-EOS. This routine automatically generates a file name based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine gdas2hdfeos (input_file, output_dir, output_file)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
```

4.1.2.5 GFS2HDFEOS

Purpose: Convert a GFS data file in GRIB format to HDF-EOS. This routine automatically generates a file name based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine gfs2hdfeos (input_file, output_dir, output_file)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
```

4.1.2.6 GOES2HDFEOS

Purpose: Convert a GOES data file in McIDAS format to HDF-EOS. This routine automatically generates a file name based on the data time. All available multispectral data matching the input file specification is combined into one multichannel HDF-EOS file. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine goes2hdfeos (input_file, output_dir, output_file, min_lat, max_lat,
min_lon, max_lon, min_time, max_time)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
real, intent (IN), optional :: min_lat                             ! minimum latitude
real, intent (IN), optional :: max_lat                             ! maximum latitude
real, intent (IN), optional :: min_lon                             ! minimum longitude
real, intent (IN), optional :: max_lon                             ! maximum longitude
character (len = 23), intent (IN), optional :: min_time            ! minimum time
character (len = 23), intent (IN), optional :: max_time            ! maximum time
```

4.1.2.7 MIRS_IMG2HDFEOS

Purpose: Convert a MIRS IMG data file in HDF format to HDF-EOS. This routine renames several of the NESDIS output arrays to conform to a consistent naming convection and removes negative algorithm diagnostics, as only one "fill_value" is employed. This routine automatically generates a file name based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine mirs_img2hdfeos (input_file, output_dir, output_file)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
integer, intent (IN) :: dataset_number                             ! dataset ID number
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
```

4.1.2.8 OLS2HDFEOS

Purpose: Convert an OLS data file in NGDC OIS format to HDF-EOS. This routine automatically generates a file name based on the data time. Since an entire OLS orbit swath is read, considerable amounts of memory may be used, although when the output_file argument is not present, the data are automatically separated into more manageable 1000 line swath sectors. However, when the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard

I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine ols2hdfeos (input_file, output_dir, output_file, min_lat, max_lat, min_lon,
max_lon, min_time, max_time)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
real, intent (IN), optional :: min_lat                             ! minimum latitude
real, intent (IN), optional :: max_lat                             ! maximum latitude
real, intent (IN), optional :: min_lon                             ! minimum longitude
real, intent (IN), optional :: max_lon                             ! maximum longitude
character (len = 23), intent (IN), optional :: min_time            ! minimum time
character (len = 23), intent (IN), optional :: max_time            ! maximum time
```

4.1.2.9 QMORPH2HDFEOS

Purpose: Convert a QMORPH 8 km data file to HDF-EOS. This routine automatically generates a file name based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine qmorph2hdfeos (input_file, output_dir, output_file)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
```

4.1.2.10 SSMI2HDFEOS

Purpose: Convert a SSM/I data file in CIRA/NVAP format or in the NESDIS HDF-EOS format to a "pure" HDF-EOS format. The NESDIS data also contain the operational NESDIS products. This routine automatically generates file names based on the data time. Since an entire day of data is read, considerable amounts of memory may be used, although when the output_file argument is not present, the data are automatically separated into smaller, more manageable individual orbital swaths. However, when the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine ssmi2hdfeos (input_file, output_dir, output_file, min_lat, max_lat,
min_lon, max_lon, min_time, max_time)

character (len = MAX_PATH), intent (IN) :: input_file       ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir ! output directory
integer, intent (IN), optional :: sat_number                ! satellite number (only
used with NESDIS input data)
```

```
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
real, intent (IN), optional :: min_lat                       ! minimum latitude
real, intent (IN), optional :: max_lat                       ! maximum latitude
real, intent (IN), optional :: min_lon                       ! minimum longitude
real, intent (IN), optional :: max_lon                       ! maximum longitude
character (len = 23), intent (IN), optional :: min_time      ! minimum time
character (len = 23), intent (IN), optional :: max_time      ! maximum time
```

4.1.2.11 SSMIS2HDFEOS

Purpose: Convert a SSMIS data file in "raw" format to HDF-EOS. This routine automatically generates file names based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine ssmis2hdfeos (input_file, output_dir, output_file, min_lat, max_lat,
min_lon, max_lon, min_time, max_time)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
real, intent (IN), optional :: min_lat                             ! minimum latitude
real, intent (IN), optional :: max_lat                             ! maximum latitude
real, intent (IN), optional :: min_lon                             ! minimum longitude
real, intent (IN), optional :: max_lon                             ! maximum longitude
character (len = 23), intent (IN), optional :: min_time            ! minimum time
character (len = 23), intent (IN), optional :: max_time            ! maximum time
```

4.1.2.12 SSMIS_EDR2HDFEOS

Purpose: Convert a SSMIS EDR data file to HDF-EOS. This code was adapted from the SSMIS2HDFEOS routine. This routine automatically generates a file name based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine ssmis_edr2hdfeos (input_file, output_dir, output_file, min_lat, max_lat,
min_lon, max_lon, min_time, max_time)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
integer, intent (IN), optional :: sat_number                       ! sat. ID number
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
real, intent (IN), optional :: min_lat                             ! minimum latitude
real, intent (IN), optional :: max_lat                             ! maximum latitude
real, intent (IN), optional :: min_lon                             ! minimum longitude
real, intent (IN), optional :: max_lon                             ! maximum longitude
character (len = 23), intent (IN), optional :: min_time            ! minimum time
character (len = 23), intent (IN), optional :: max_time            ! maximum time
```

4.1.2.13 SSMT22HDFEOS

Purpose: Convert a SSM/T-2 data file in NESDIS Level 1B format to HDF-EOS. This routine automatically generates file names based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine ssmt22hdfeos (input_file, output_dir, output_file, min_lat, max_lat,
min_lon, max_lon, min_time, max_time)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
real, intent (IN), optional :: min_lat                             ! minimum latitude
real, intent (IN), optional :: max_lat                             ! maximum latitude
real, intent (IN), optional :: min_lon                             ! minimum longitude
real, intent (IN), optional :: max_lon                             ! maximum longitude
character (len = 23), intent (IN), optional :: min_time            ! minimum time
character (len = 23), intent (IN), optional :: max_time            ! maximum time
```

4.1.2.14 TMI2HDFEOS

Purpose: Convert a TRMM TMI data file in HDF Level 1B format to HDF-EOS. This routine automatically generates file names based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine tmi2hdfeos (input_file, output_dir, output_file, min_lat, max_lat, min_lon,
max_lon, min_time, max_time)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
real, intent (IN), optional :: min_lat                             ! minimum latitude
real, intent (IN), optional :: max_lat                             ! maximum latitude
real, intent (IN), optional :: min_lon                             ! minimum longitude
real, intent (IN), optional :: max_lon                             ! maximum longitude
character (len = 23), intent (IN), optional :: min_time            ! minimum time
character (len = 23), intent (IN), optional :: max_time            ! maximum time
```

4.1.2.15 VIRS2HDFEOS

Purpose: Convert a TRMM VIRS data file in HDF Level 1B format to HDF-EOS. This routine automatically generates file names based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is

specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine virs2hdfeos (input_file, output_dir, output_file, min_lat, max_lat,
min_lon, max_lon, min_time, max_time)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
real, intent (IN), optional :: min_lat                             ! minimum latitude
real, intent (IN), optional :: max_lat                             ! maximum latitude
real, intent (IN), optional :: min_lon                             ! minimum longitude
real, intent (IN), optional :: max_lon                             ! maximum longitude
character (len = 23), intent (IN), optional :: min_time            ! minimum time
character (len = 23), intent (IN), optional :: max_time            ! maximum time
```

4.1.2.16 WINDSAT2HDFEOS

Purpose: Convert a WindSat data file to HDF-EOS. This routine automatically generates file names based on the data time. When the output_file argument is present, the output file name is returned and points to the internal HDF-EOS data structure that has been created (but not written to disk; i.e., no hard I/O is performed when output_file is specified). If called with the output_file argument, the created HDF-EOS data structure remains after the call to this routine and it is the caller's responsibility to deallocate the created HDF-EOS data structure.

```
subroutine virs2hdfeos (input_file, output_dir, output_file, min_lat, max_lat,
min_lon, max_lon, min_time, max_time)

character (len = MAX_PATH), intent (IN) :: input_file              ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir              ! output directory
character (len = MAX_PATH), intent (OUT), optional :: output_file  ! output file
real, intent (IN), optional :: min_lat                             ! minimum latitude
real, intent (IN), optional :: max_lat                             ! maximum latitude
real, intent (IN), optional :: min_lon                             ! minimum longitude
real, intent (IN), optional :: max_lon                             ! maximum longitude
character (len = 23), intent (IN), optional :: min_time            ! minimum time
character (len = 23), intent (IN), optional :: max_time            ! maximum time
```

4.1.3 Application Specific Routines

Application specific routines are unique user-written subroutines found in script files. They tend to be customized applications and may not be suitable for use by all users without a more detailed understanding of the particular application. The routines below are merely examples of such application specific subroutines.

4.1.3.1 AMSU2McIDAS

Purpose: This routine converts an AMSU HDF-EOS data structure into a series of McIDAS formatted files. Examine the source codes and external documentation materials for more details.

```
subroutine amsu2mcidas (input_file, output_dir, tbus_dir)

character (len = MAX_PATH), intent (IN) :: input_file       ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir       ! output directory name
character (len = MAX_PATH), intent (IN) :: tbus_dir         ! TBUS directory name
```

4.1.3.2 Apply_RR_Correction

Purpose: This routine applies the rainfall rate corrections. Note: This routine reads the ASCII RR correction statistics files created by the `rr_statistics` routine. Strength inputs are 0 = no correction, 1 = light correction, 2= full correction.

```
subroutine apply_rr_correction (input_file, correction_dir, ocean_ref_sat,
      ocean_strength, land_ref_sat, land_strength, output_dir, output_file)

character (len = MAX_PATH), intent (IN) :: input_file       ! input file name
character (len = MAX_PATH), intent (IN) :: correction_dir   ! correction directory
character (len = MAX_PATH), intent (IN) :: ocean_ref_sat    ! ocean ref. satellite
integer, intent (IN) :: ocean_strength                      ! ocean strength
character (len = MAX_PATH), intent (IN) :: land_ref_sat     ! land ref. satellite
integer, intent (IN) :: land_strength                       ! land strength
character (len = MAX_PATH), intent (IN) :: output_dir       ! output directory
character (len = MAX_PATH), intent (OUT) :: output_file     ! output file name
```

4.1.3.3 Apply_TPW_Correction

Purpose: This routine applies the total precipitable water corrections. Note: This routine reads the ASCII TPW correction statistics files created by the `create_tpw_correction` routine.

```
subroutine apply_tpw_correction (input_file, correction_dir, output_dir, output_file)

character (len = MAX_PATH), intent (IN) :: input_file       ! input file name
character (len = MAX_PATH), intent (IN) :: correction_dir   ! correction directory
character (len = MAX_PATH), intent (IN) :: output_dir       ! output directory
character (len = MAX_PATH), intent (OUT) :: output_file     ! output file name
```

4.1.3.4 Bsmooth

Purpose: This routine smooths a 2-D real number data field in an HDF-EOS grid file using a binomial filter. All I/O is to and from the hard drive.

```
subroutine bsmooth (input_file, output_file, field_name, filter_size)

character (len = MAX_PATH), intent (IN) :: input_file       ! input file name
character (len = MAX_PATH), intent (OUT) :: output_file     ! output file name
character (len = MAX_PATH), intent (IN) :: field_name       ! field name
integer, intent (IN) :: filter_size                         ! filter size
```

4.1.3.5 BTPW_Land_Blend

Purpose: This routine blends GPS, MIRS, and GOES Sounder data with the Blended Total Precipitable Water product. The blending is performed in multiple stages. Examine the source codes and PSDI documentation materials for more details.

Currently science version 2 is supported.

```
subroutine btpw_land_blend (version_science, input_merged_tpw_file, output_merged_tpw_file,
       gps_ascii_data_file, gps_station_file, goes_east_ascii_file, goes_west_ascii_file,
       mirs_surface_file, diagnostics_file)

integer, intent (IN) :: version_science                          ! science version
character (len = MAX_PATH), intent (IN) :: input_merged_tpw_file   ! input file name
character (len = MAX_PATH), intent (OUT) :: output_merged_tpw_file ! output file name
character (len = MAX_PATH), intent (IN) :: gps_ascii_data_file     ! GPS file name
character (len = MAX_PATH), intent (IN) :: gps_station_file        ! GPS station file name
real, intent (IN) :: max_reasonable_tpw                          ! max. TPW value
character (len = MAX_PATH), intent (IN) :: goes_east_ascii_file    ! GOES-E file name
character (len = MAX_PATH), intent (IN) :: goes_west_ascii_file    ! GOES-W file name
character (len = MAX_PATH), intent (IN) :: mirs_surface_file       ! MIRS surface file name
character (len = MAX_PATH), intent (IN) :: diagnostics_file        ! diagnostics file name
```

4.1.3.6 Copy_NESDIS_TBUS_File

Purpose: This routine copies a NESDIS TBUS file and renames it to conform to CIRA naming conventions. This routine expects the new TBUS file to be called `<tbus_dir>tbus.new` and renames the file to `<tbus_dir>NESDIS\tbus.yyddd` where `yyddd` are determined from the file contents.

```
subroutine copy_nesdis_tbus_file (tbus_dir)

character (len = MAX_PATH), intent (IN) :: tbus_dir  ! output TBUS directory name
```

4.1.3.7 Compute_TPW_Percent

Purpose: This routine computes the TPW percent of weekly NVAP normal (1988-1999). Examine the source codes and external documentation materials for more details.

```
subroutine compute_tpw_percent (merged_tpw_file, nvap_mean_file, output_pct_file)

character (len = MAX_PATH), intent (IN) :: merged_tpw_file  ! input file name
character (len = MAX_PATH), intent (IN) :: nvap_mean_file   ! NVAP mean file name
character (len = MAX_PATH), intent (OUT) :: output_pct_file ! output file name
```

4.1.3.8 Create_TPW_Correction

Purpose: This routine creates the total precipitable water pentad correction statistics. Note: This routine creates the ASCII TPW correction statistics files used by the `apply_tpw_correction` routine.

```
subroutine create_tpw_correction (input_file_spec, output_dir)
```

```
character (len = MAX_PATH), intent (IN) :: input_file_spec  ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir       ! output directory
```

4.1.3.9 GOES_SNDR_TPW_Merge

Purpose: This routine reads the CIMSS GOES Sounder ASCII data files and merges the Blended TPW product with the GOES Sounder data. Examine the source codes and external documentation materials for more details.

```
subroutine goes_sndr_tpw_merge (output_merged_tpw_file, goes_ascii_data_file)

character (len = MAX_PATH), intent (IN) :: output_merged_tpw_file  ! input file name
character (len = MAX_PATH), intent (IN) :: gps_ascii_data_file     ! GPS file name
```

4.1.3.10 GPS_TPW_Interpolate

Purpose: This routine reads the NOAA/ESRL/GSD GPS ASCII data files and merges the Blended TPW product with the GOES Sounder data. Examine the source codes and external documentation materials for more details.

```
subroutine gps_tpw_interpolate (merged_tpw_file, gps_ascii_data_file, gps_station_file,
       output_gps_merge_file, max_reasonable_tpw, landmask_file)

character (len = MAX_PATH), intent (IN) :: merged_tpw_file         ! input file name
character (len = MAX_PATH), intent (IN) :: gps_ascii_data_file     ! GPS file name
character (len = MAX_PATH), intent (IN) :: gps_station_file        ! GPS station file name
character (len = MAX_PATH), intent (OUT) :: output_gps_merge_file  ! output file name
real, intent (IN), optional :: max_reasonable_tpw                  ! max. TPW value
character (len = MAX_PATH), intent (IN) :: goes_east_ascii_file    ! GOES-E file name
character (len = MAX_PATH), intent (IN) :: goes_west_ascii_file    ! GOES-W file name
character (len = MAX_PATH), intent (IN), optional :: landmask_file ! landmask file
```

4.1.3.11 RR_Statistics

Purpose: This routine creates the rainfall rate correction statistics. Note: This routine creates the ASCII RR correction statistics files used by the `apply_rr_correction` routine.

```
subroutine rr_statistics (input_file_spec, output_dir)

character (len = MAX_PATH), intent (IN) :: input_file_spec  ! input file name
character (len = MAX_PATH), intent (IN) :: output_dir       ! output directory
```

4.1.3.12 Write_McIDAS

Purpose: This routine writes a DPEAS data structure data item to a McIDAS format data file.

```
subroutine write_mcidas (mapped_file, output_parameter, output_file, timesat, smooth)

character (len = MAX_PATH), intent (IN) :: mapped_file      ! input HDF-EOS file name
character (len = MAX_PATH), intent (IN) :: output_parameter ! output data item name
character (len = MAX_PATH), intent (IN) :: output_file      ! output file name
logical, intent (IN) :: timesat                        ! time and satellite, if .TRUE.
integer, intent (IN) :: smooth                         ! smoothing factor
```

4.2 Key DPEAS Structures

4.2.1 DPEAS Memory Data Structure

The DPEAS memory data structure is a series of arrays that mimic the HDF-EOS data format design. It is only present in the memory of the machines and does not exist as a file on the drive. The memory is referenced via a series of Fortran pointers. The top-level pointers are referred by their virtual file names. Thus, the structure looks like an imaginary hard drive. This allows data fields and a variety of complex elements to be referenced by the various generalized routines.

The easiest way to explore the DPEAS memory data structure is to view it with the DPEAS intrinsic routine `DPE_WRITE_DATA_STRUCTURE`. This routine will dump the contents of the memory structure (but not the data) to the DPEAS output. This is very useful for viewing and verifying that new data sets have been read into DPEAS correctly. A good HDF-EOS viewer or browser will perform the same functionality.

A freeware HDF-EOS browser is included on the DPEAS CD-R under the `\DPEAS files\` directory. Also included under that directory are many HDF-EOS file examples, which you can browse and examine using the DPEAS tools. To read a "pure" HDF-EOS file (see Jones and Vonder Haar, 2002) within DPEAS, use the `READ_HDFEOS` DPEAS intrinsic routine.

The full description of the DPEAS memory data structure is beyond the scope of this document. However, it is discussed in the companion technical document, *DPEAS Programmer's Guide,* for those who are interested or require a better understanding. A definitive source of HDF-EOS information is the *HDF-EOS Library User's Guide for the ECS project, Volume 1: Overview and Examples* (http://edhs1.gsfc.nasa.gov/waisdata/sdp/pdf/tp17060001.pdf).

4.2.2 DPEAS Directory Structure

The following DPEAS subdirectories are necessary for DPEAS to function correctly. Do not attempt to run DPEAS without these directories:

DPEAS\bin

The bin directory contains compiled executables.

DPEAS\data

The data directory contains input and output data files, as well as DPEAS processing state information (see comments below related to the DPEAS\data\work directory).

DPEAS\data\input

This is the main user input directory. It is suggested (but not required) that all user-created DPEAS input script files should reside in this directory. The input files are ASCII text files and thus are editable by any common ASCII file editor (e.g., NOTEPAD). A common error is to assume that the input files are Fortran source files and are therefore in need of compilation. This is not the case. The input files are merely interpreted ASCII script files that

use the familiar Fortran 90 syntax. This should allow users (at least those with some Fortran background) to have instant familiarity with the DPEAS user interface. DPEAS uses pseudo-Fortran code as its input file. This does not mean it needs a compiler. DPEAS will interpret the code in real-time, skipping the normal compilation step. This results in a multitude of benefits, which will become apparent with use of the DPEAS system.

DPEAS\data\output

This is the main user output directory. Its use is optional, as other data directory organization schemes could be used instead.

DPEAS\data\work

The work directory is used by DPEAS to write temporary working files or current process state information. Normally, DPEAS cleans up these files as needed so that no user intervention is needed.

DPEAS\logs

The logs directory contains all log files.

DPEAS\monitoring

The monitoring directory contains files related to the monitoring program.

DPEAS\scripts

The scripts directory contains all script files.

DPEAS\setup

The setup directory contains configuration and setup files.

DPEAS\setup\configuration\build

The build directory is used to store information regarding the DPEAS build version. Users should not modify this directory. DPEAS updates these files as needed. No user intervention is necessary.

DPEAS\setup\configuration\database

The database directory is used by DPEAS to write the DPEAS database files for each DPEAS node. Users should not modify this directory. DPEAS updates these files as needed. No user intervention is necessary.

DPEAS\setup\configuration\resource

The resource directory is used to specify available nodes for use in DPEAS processing. Each DPEAS node should have its own resource file. The resource files are ASCII text files. They can be modified at any time, even while DPEAS is executing (See section 3.3.2 for information about the resource file configuration).

DPEAS\src

The src directory contains all source codes. The current DPEAS Windows source distribution exists under the DPEAS\src\DPEAS subdirectory.

4.3 DPEAS Parallel Processing

DPEAS is capable of running on several computers (nodes) simultaneously. DPEAS implements the multinode processing with a MASTER/SLAVE paradigm. Resource files are used to specify attributes of the master and slave nodes. The node on which the DPEAS executable resides is the master node. The master node delegates data processing chores to the DPEAS slave nodes. Multinode processing is implemented using an "independent DO loop construct", an idea borrowed from High Performance Fortran (HPF). Primary level DO loops in a DPEAS master input file are intrinsically assumed to be independent DO loops, thus iterations of the DO loop may be performed in any order. The designated slave node processes remaining nested DO loops. Inter-process communication is prohibited between the independent DO loop block and the host DPEAS program variables. Thus, variable values modified within an independent DO loop are not available to the host DPEAS program. To turn off the automatic parallelization of DPEAS input, add `call DPE_SLAVE` at the start of the execution part of the input file. An example of the DPEAS data processing flow is shown below. Each block is a group of Fortran 90 statements that are interpreted during DPEAS execution

4.3.1 Running DPEAS in Parallel Mode

Assuming that you've configured DPEAS and BJS as described in Section 3.3, submit your DPEAS batch file to BJS for execution with the argument containing the relative path name to the input DPEAS script file from the DPEAS executable.

Fig. 1 illustrates the DPEAS parallelism and Fig. 2 shows how to monitor parallel mode DPEAS runs with the BJS Client. Fig. 3 contains a high-level schematic of the DPEAS data processing flow as the DPEAS Fortran interpreter automatically segments the main input into subtasks.

Figure 1: DPEAS Parallelism

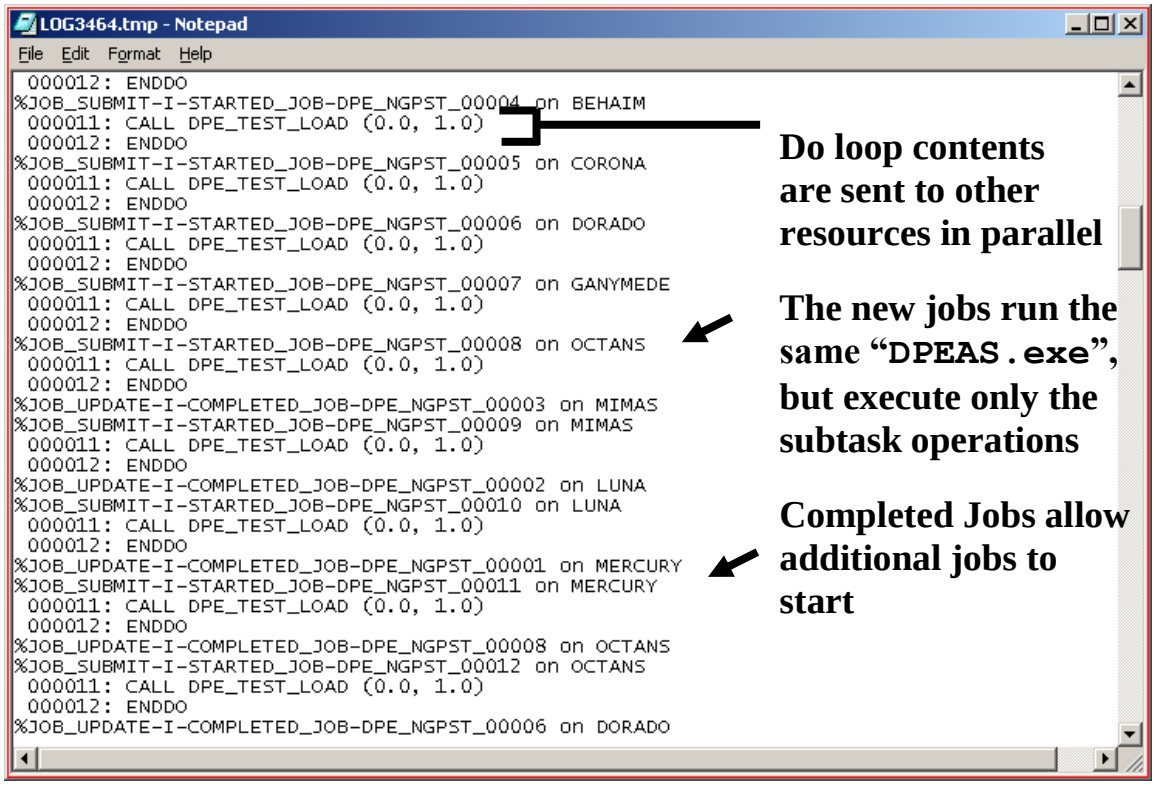


Figure 2: Monitoring Parallel Mode DPEAS Runs with the Batch Job Server Client

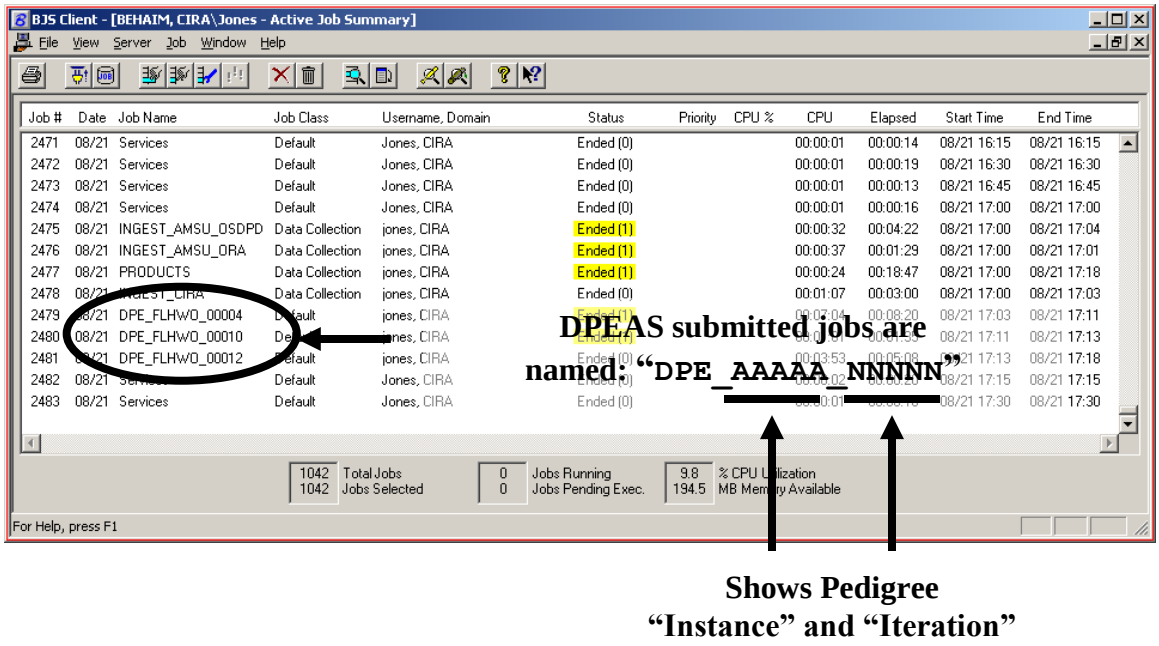
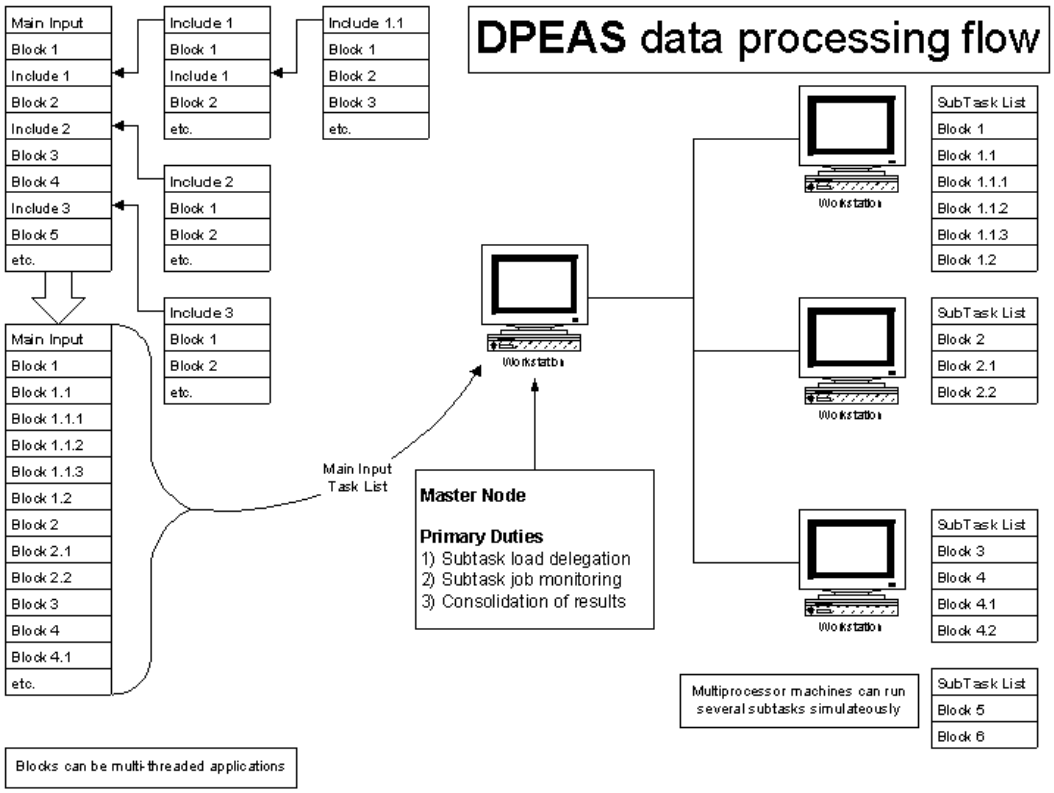


Figure 3: DPEAS Data Processing Flow



4.3.2 Aborting a Parallel Mode Job with the Batch Job Server Client

To abort a parallel mode job manually:

1. Cancel the Master job
2. Allow Spawned jobs to run and complete normally or cancel them manually (this works even if they were submitted by another user)
    a. Examine the Spawned job names (e.g. "DPE_AAAAA_nnnnn")
    b. Use the BJS Client to connect to the specified machine
    c. Cancel the remaining spawned jobs (remember, they may have finished by the time you get to them)
    d. If canceling another user's job, his or her master job will continue to wait until the canceled job is manually restarted on another system. Therefore, it is imperative that users communicate with each other when canceling another user's master job.

4.4 Log Files

DPEAS directs all message output to the standard output device (normally the DOS command window). When running in batch mode, this output is captured as a log file. The BJS client can

browse all DPEAS log files interactively and is very useful when exploring the DPEAS output results.

4.4.1 File I/O Messages

To determine what fortran file I/O is used in DPEAS, search for the DPEAS error messages containing "**%SYS_OPEN-I-FILE**-"<filename>", **ACTION =** <action_status>".

In addition the DPEAS C/C++ I/O libraries track their file I/O by the "<PROCEDURE_NAME>-I-**FILE**-<filename>" error message, or through more explicit "<PROCEDURE_NAME>-I-**INPUT_FILE**-<filename> ", and "<PROCEDURE_NAME>-I-**OUTPUT_FILE**-<filename>", error messages.

The DPEAS data mirroring facility issues its own error messages, that start with the string pattern "%SYS_MIRROR*".

Other DPEAS file I/O error messages denote particular processing states that are dependent on the specific DPEAS routine.

4.4.2 Error Handling

DPEAS attempts to handle abnormal terminations and will exit with the appropriate status automatically. The error levels in Table 1 are recognized.

Table 1: DPEAS Error Levels

ID	Error Level	Behavior
S	Success	DPEAS continues
I	Informational	DPEAS continues
W	Warnings	DPEAS continues
R	Return	DPEAS returns from the current subtask and continues
E	Errors	DPEAS conditionally terminates (depending upon the processing context)
F	Fatal Errors	DPEAS terminates

5 DPEAS ADVANCED FEATURES

5.1 Parallelism

DPEAS has been tested in a 20-processor configuration, and simulation tests have shown that it is capable of scaling to approximately 2000 processors with current hardware capabilities available at CIRA. Parallelization capabilities are performed by dynamically propagating the executable to additional computing nodes as needed. Thus, the system replicates itself as needed and reports to the host program when program segments are complete. This is automated within DPEAS, and the user code is distributed easily, since it is part of the DPEAS executable. The replication process is controlled by configuration data in simple text files. Asynchronous and independent parallel DO loop constructs are supported in the input script semantics, thus allowing for easy parallelization. They are controlled by simple input commands that switch the parallelization mode to synchronous or asynchronous behaviors. This allows a user to simply flag a section that requires synchronous behavior.

5.1.1 Synchronous Parallelism

By default, DPEAS submits jobs in synchronous mode. That is, each parallel program segment (e.g. DO loop block) is completed in sequence, with jobs in the next program segment pending until the current parallel program segment is completed. This behavior can also be invoked by calling the `dpe_mode_synchronous` routine directly (more information is available in section 4.1.1.8).

5.1.2 Asynchronous Parallelism

DPEAS is also capable of asynchronous parallelism behaviors. In asynchronous mode, each program segment is performed independently and without regard to the completion of the previous program segment. This mode assumes that each program segment is fully independent of all other program segments. The asynchronous mode only resynchronizes at the end of the master script or when invoked by calling the `dpe_mode_asynchronous` routine (more information is available in section 4.1.1.7).

5.2 Data Mirroring

The DPEAS data mirroring capabilities are provided by the DPEAS near-real time replication file subsystem (NRTRFS). The NRTFRS is necessary for continuance of the data integrity within a cluster during failover and recovery operations.

The system is tightly integrated with the HDF-EOS Virtual I/O Subsystem (HVIOS) and includes a user-callable interface. The interface overloads the system-dependent file open and close statements with the file replication capabilities of the NRTRFS. The file replication actions are constrained by the time and date that the file was last modified. For example, if a source file's last modification time is given by *S*, and if the replication target file's modification time (if

it exists) is given by T, the cases in Table 2 apply. In table 2, $\Delta = T - S$ and the defaults for MIN and MAX are –12 and 0 h, respectively. This constructs a temporal window of opportunity that designates when replication will be performed. The MIN constraint is set to $-\infty$ to perform a complete replication cycle. The constraint limits the consumption of resources by the NRTRFS and allows DPEAS to continue processing with minimal downtime after a failover event.

Table 2: DPEAS Data Mirroring as a Function of the Replication Target File's Modification Time

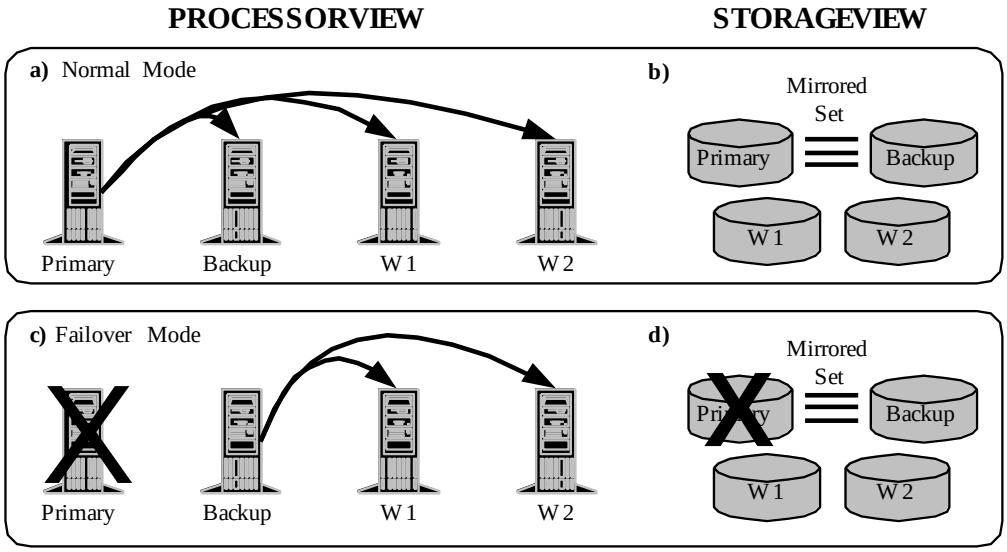
Case T_1	No operation	$\Delta <$ MIN
Case T_2	Replicate	MIN $\geq \Delta >$ MAX
Case T_3	No operation	$\Delta \geq$ MAX

The replication configuration is contained in ASCII text files for easy modification and use. It is possible to create a distributed peer-to-peer (P2P) computing cluster in DPEAS that would continue to function even as each node in the cluster was powered off. The P2P nature of DPEAS would simply reroute the formerly active parallel tasks on the down nodes to other computing nodes, and continue on, making use of the replicated data sources that are maintained via the NRTRFS. On recovery, the NRTRFS will reconstruct the replicated drives. If a particular node has been down for a long time, a complete replication cycle would need to be initiated manually. This can occur in parallel with concurrent operational computing tasks, since file locking mechanisms are built into the DPEAS architecture.

5.3 Fault Resilience

DPEAS is capable of running in a fully mirrored configuration with a primary and backup system (Figure 4). This provides failover capabilities should the primary system experience a hardware-related failure. The failover procedure is fully automated so that data operations continue in the event of a hardware (e.g. disk drive) failure. The backup system is a computer that otherwise serves as a normal worker node in the DPEAS parallel cluster. Restoration of the system is manually initiated. Movement of the DPEAS system between the primary and backup systems is facilitated by the DPEAS system state information, which is mirrored to the OS file system of the backup system. The failover function is performed by specialized static scripts that work in coordination with the DPEAS System State Manager (SSM). The DPEAS SSM maintains and updates the DPEAS system state information. The DPEAS system state information is stored in simple ASCII text files on the primary system and is usually replicated to other nodes to ensure system reliability in the event of a hardware failure. In addition to the hardware failover capabilities, various software error states are monitored so that DPEAS can shut down errant processes on remote nodes without user intervention.

Figure 4: Normal Mode of the DPEAS Hardware Configuration



Failover Steps
(Automated)

1. Synchronize states
2. Promote the Backup

Restore Steps
(Manual)

1. Demote the Backup
2. Synchronize states
3. Reactivate Primary

a) The normal mode of the DPEAS hardware configuration is such that the primary node can submit processing subtasks to any worker node in a multiple cluster environment.
b) The backup system routinely mirrors the primary system state. This makes the system resilient to hardware failures. After a primary system failure the subsystem goes into failover mode.
c) In failover mode, automated DPEAS failover procedures promote the backup node to take over the remaining processing duties.
d) Also in failover, the NRTRFS automatically redirects all I/O requests to the mirrored files. The DPEAS system state is synchronized before promotion and demotion of the DPEAS coordinating node, allowing a controlled failover to occur.

6 APPENDIX A: "A DYNAMIC PARALLEL DATA-COMPUTING ENVIRONMENT FOR CROSS-SENSOR SATELLITE DATA MERGER AND SCIENTIFIC ANALYSIS" APPENDIX A: "A DYNAMIC PARALLEL DATA-COMPUTING ENVIRONMENT FOR CROSS-SENSOR SATELLITE DATA MERGER AND SCIENTIFIC ANALYSIS"

7 ACKNOWLEDGEMENTS

The original edition (January 2005) of this DPEAS User's guide was created by Katherine P. Kidder, Andrew S. Jones and Stanley Q. Kidder. It has been updated to reflect the DPEAS 2.x and 3.x modifications.