Data Processing and Error Analysis System (DPEAS)

Programmer’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

TABLE OF CONTENTS ................................................................................................................. I  
LIST OF FIGURES ....................................................................................................................... V  
LIST OF TABLES .......................................................................................................................... V  
LIST OF ABBREVIATIONS .......................................................................................................... VI  
1 INTRODUCTION ...................................................................................................................... 1  
2 OVERVIEW OF THIS DOCUMENT .......................................................................................... 2  
  2.1 DPEAS VERSIONS .......................................................................................................... 2  
3 INSTALLATION AND CONFIGURATION .................................................................................. 3  
  3.1 DPEAS SYSTEM REQUIREMENTS .............................................................................. 3  
  3.2 INSTALLING DPEAS .................................................................................................. 3  
  3.3 CONFIGURING DPEAS ............................................................................................... 4  
    3.3.1 Batch Job Server (BJS) Setup ........................................................................... 4  
    3.3.2 Resource File Configuration ....................................................................... 5  
  3.4 DPEAS COMPILER SETTINGS ..................................................................................... 6  
    3.4.1 Known Compiler Bugs and Possible Recompilation Issues ....................... 7  
    3.4.2 Current Compiler Settings ........................................................................... 7  
    3.4.2.1 Current Debug Options ........................................................................ 7  
    3.4.2.2 Current Release Options ..................................................................... 8  
    3.4.2.3 Current Library Options ................................................................. 8  
    3.4.3 Older Compiler Settings (for reference) ..................................................... 9  
    3.4.3.1 Debug Options .................................................................................... 9  
    3.4.3.2 Release Options .................................................................................. 10  
    3.4.3.3 Library Options .................................................................................. 10  
  3.5 UPDATING THE DPEAS COMPILER TOOL SETTINGS ......................................... 11  
4 PROGRAMMING DPEAS (TBR) ............................................................................................ 12  
  4.1 MS VISUAL STUDIO PRIMER ................................................................................... 13  
    4.1.1 Running in Debug Mode with the MS Visual Studio Environment .......... 13  
    4.1.2 Running in Release Mode with the MS Visual Studio Environment .... 13  
  4.2 BATCH JOB SERVER PRIMER ................................................................................. 13  
  4.3 MODIFYING DPEAS ..................................................................................................... 13  
    4.3.1 The 3 Programming Steps to Add a User Routine to DPEAS .................. 14  
    4.3.2 Things to Remember ................................................................................. 14  
    4.3.3 Examples .................................................................................................... 15  
ANOTHER EXAMPLE OF INTEGRATING A ROUTINE INTO THE DPEAS SOFTWARE CAN BE FOUND IN APPENDIX C. .................................................................................. 22  
5 DPEAS MODULES .................................................................................................................. 23  
  5.1 CREATING A NEW DPEAS APPLICATION ROUTINE .................................... 23
5.1.1 Example ........................................................................................................... 23
5.2 Creating a New Root-Level Module ................................................................. 24

6 APPENDIX A: “A DYNAMIC PARALLEL DATA-COMPUTING ENVIRONMENT FOR CROSS-SENSOR SATellite DATA MERGER AND SCIENTIFIC ANALYSIS” ................................................................. 25

7 APPENDIX B: DPEAS MODULES ...................................................................... 26

7.1 COMMON: CONSTANTS_MODULE ......................................................................... 26
7.2 COMMON: DPEAS_MODULE .................................................................................. 26
7.3 COMMON: IO_MODULE ......................................................................................... 26
7.4 COMMON: MATH_MODULE .................................................................................. 26
7.5 COMMON: PHYSICS_MODULE .............................................................................. 27
7.6 COMMON: RANGE_MODULE ................................................................................. 27
7.7 COMMON: SECTOR_MODULE ................................................................................ 27
7.8 COMMON: TIME_MODULE .................................................................................... 28
7.9 DATA: DPE_DATA_ALLOCATE_MODULE ............................................................. 30
7.10 DATA: DPE_DATA_ASSIGNMENT_11_MODULE ............................................... 30
7.11 DATA: DPE_DATA_ASSIGNMENT_12_MODULE ............................................... 30
7.12 DATA: DPE_DATA_ASSIGNMENT_14_MODULE ............................................... 31
7.13 DATA: DPE_DATA_ASSIGNMENT_MODULE ....................................................... 31
7.14 DATA: DPE_DATA_ASSIGNMENT_R4_MODULE ............................................... 31
7.15 DATA: DPE_DATA_ASSIGNMENT_R8_MODULE ............................................... 31
7.16 DATA: DPE_DATA_COPY_MODULE ..................................................................... 32
7.17 DATA: DPE_DATA_CREATE_MODULE .................................................................. 32
7.18 DATA: DPE_DATA_DEALLOCATE_MODULE ....................................................... 32
7.19 DATA: DPE_DATA_FIND_MODULE ..................................................................... 32
7.20 DATA: DPE_DATA_FOUND_MODULE ................................................................... 33
7.21 DATA: DPE_DATA_HDFEOS_GDREAD_MODULE ............................................ 33
7.22 DATA: DPE_DATA_HDFEOS_GDWRITE_MODULE ........................................... 33
7.23 DATA: DPE_DATA_HDFEOS_PTRREAD_MODULE .......................................... 33
7.24 DATA: DPE_DATA_HDFEOS_PTRWRITE_MODULE .......................................... 33
7.25 DATA: DPE_DATA_HDFEOS_READ_MODULE .................................................... 34
7.26 DATA: DPE_DATA_HDFEOS_SWRITE_MODULE .............................................. 34
7.27 DATA: DPE_DATA_HDFEOS_SWRITE_MODULE .............................................. 34
7.28 DATA: DPE_DATA_HDFEOSUTILITY_MODULE .............................................. 34
7.29 DATA: DPE_DATA_HDFEOS_WRITE_MODULE ............................................... 35
7.30 DATA: DPE_DATA_MODULE .............................................................................. 35
7.31 DATA: DPE_DATA_STRUCTURE_MODULE .......................................................... 35
7.32 DATA: DPE_DATA_Utility_MODULE .................................................................. 35
7.33 DPE: DPE_ARGUMENT_MODULE ...................................................................... 36
7.34 DPE: DPE_COMMON_MODULE .......................................................................... 36
7.35 DPE: DPE_DATABASE_MODULE ........................................................................ 37
7.36 DPE: DPE_Utility_MODULE .............................................................................. 37
7.37 DPE: DPE_VARIABLE_MODULE ....................................................................... 38
7.38 ERRORS: ERROR_MESSAGE (FACILITY, LEVEL, IDENTIFICATION, MESSAGE) ................................................................. 38
7.39 ERRORS: ERROR_MODULE .............................................................................. 38
7.86 TRANSLATORS: SSMIS_EDR2HDFEOS_MODULE ................................................................. 52
7.87 TRANSLATORS: SSMR2HDFEOS_MODULE ......................................................................... 52
7.88 TRANSLATORS: TM3HDFEOS_MODULE .............................................................................. 52
7.89 TRANSLATORS: VIRS2HDFEOS_MODULE ........................................................................... 52
7.90 TRANSLATORS: WINDSAT2HDFEOS_MODULE ................................................................. 53
7.91 NESDIS: LANDEM_V2_MODULE ......................................................................................... 53
7.92 NESDIS: LANDEM_V3_MODULE ......................................................................................... 53
7.93 NESDIS: NESDIS_OCEANEM_Module ................................................................................ 53
7.94 NESDIS: TYPE_KINDS ..................................................................................................... 54
7.95 NESDIS: WENG_EMISS_MODEL ....................................................................................... 54
7.96 OSPO/FORSYTHE: BTPW_LAND_BLEND_MODULE .......................................................... 54
7.97 OSPO/FORSYTHE: BTPW_LAND_BLEND_V2_MODULE ......................................................... 54
7.98 OSPO/FORSYTHE: CIMSS_GEOS_SOUNDER_MODULE ..................................................... 55
7.99 OSPO/FORSYTHE: COMPUTE_TPW_PERCENT_MODULE ................................................. 55
7.100 OSPO/FORSYTHE: FORSYTHE_UTILITY_MODULE ......................................................... 55
7.101 OSPO/FORSYTHE: GPS_TPW_INTERPOLATE_MODULE ..................................................... 55
7.102 OSPO/JONES: CCDA_MODULE .......................................................................................... 56
7.103 OSPO/KIDDER: BINOMIAL_SMOOTHER_MODULE .......................................................... 56
7.104 OSPO/KIDDER: SQK_MCIDAS_MODULE ......................................................................... 56
7.105 OSPO/KIDDER: SQK_PENTAD_CORRECTION_MODULE .................................................. 56
7.106 OSPO/KIDDER: SQK_RR_CORRECTION_MODULE ............................................................ 57
7.107 APPS: DPE_APPLICATIONS_MODULE ................................................................................ 57
7.108 COMBS: COMBS_MODULE ................................................................................................ 57
7.109 FORSYTHE: FORSYTHE_MODULE ................................................................................ 57
7.110 KIDDER: KIDDER_MODULE ............................................................................................ 58
7.111 USER: DPE_USER_MODULE ............................................................................................. 58
7.112 USER: EXAMPLE_EMISSIVITY_MODULE ....................................................................... 58
7.113 JONES: JONES_MODULE ................................................................................................ 58
7.114 JONES: CCDA_MODULE ................................................................................................... 59
7.115 TEST: TEST_CALL_MODULE ............................................................................................ 59
7.116 TEST: TEST_HDFEOS_MODULE ......................................................................................... 59
7.117 TEST: TEST_KINDS_MODULE ........................................................................................... 59
7.118 TEST: TEST_LIST_MODULE ............................................................................................. 59
7.119 TEST: TEST_LOAD_MODULE ............................................................................................. 60
7.120 TEST: TEST_MODULE ....................................................................................................... 60
7.121 TEST: TEST_STRING_MODULE .......................................................................................... 60
7.122 DPEAS: DPE_INPUT_MODULE ........................................................................................ 60
7.123 DPEAS: DPE_MODULE ..................................................................................................... 61
7.124 DPEAS: DPE_SYNTAX_MODULE ....................................................................................... 61
7.125 DPEAS: DPE_VERSION .................................................................................................. 61
7.126 DPEAS: DPEAS .............................................................................................................. 61

8 APPENDIX C: DPEAS TUTORIAL .......................................................................................... 62

9 ACKNOWLEDGEMENTS ..................................................................................................... 72
LIST OF FIGURES

FIGURE 1: THE RELATION OF THE “USER_MODULE.f90” TO DPEAS INPUT SCRIPTS ........................................ 12
FIGURE 2: THE USER’S PROGRAM HOOK ........................................................................................................... 16
FIGURE 3: THE USER’S WRAPPER ROUTINE ....................................................................................................... 17
FIGURE 4: THE USER’S APPLICATION ROUTINE DECLARATIONS (1 OF 2) ......................................................... 18
FIGURE 5: THE USER’S APPLICATION ROUTINE DECLARATIONS (2 OF 2) ......................................................... 18
FIGURE 6: THE USER’S APPLICATION ROUTINE USING EXISTING VIRTUAL I/O DATA STRUCTURES ............. 19
FIGURE 7: THE USER’S APPLICATION ROUTINE CREATING NEW VIRTUAL I/O DATA STRUCTURES ........... 19
FIGURE 8: THE USER’S APPLICATION ROUTINE USING THE VIRTUAL I/O DATA VIA POINTERS ................. 20
FIGURE 9: USAGE OF THE NEW USER ROUTINE IN A DPEAS INPUT SCRIPT FILE .................................... 20
FIGURE 10: THE RESULTS: COMPLETE INTEGRATION ...................................................................................... 21
FIGURE 11: THE OUTPUT HDF-EOS FILE ............................................................................................................ 21
FIGURE 12: THE OUTPUT IMAGE REPRESENTATION ........................................................................................... 22

LIST OF TABLES

TABLE 1: RECOGNIZABLE ERROR LEVELS ............................................................................................................ 38
TABLE 2: CALIBRATION DOCUMENTATION MATRIX .......................................................................................... 49
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMSU</td>
<td>Advanced Microwave Sounding Unit</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>BJS</td>
<td>Batch Job Server</td>
</tr>
<tr>
<td>CD-R</td>
<td>Compact Disk-Recordable</td>
</tr>
<tr>
<td>CIRA</td>
<td>Cooperative Institute for Research in the Atmosphere</td>
</tr>
<tr>
<td>CLIB</td>
<td>Cross-Sensor Processing Environment</td>
</tr>
<tr>
<td>CPE</td>
<td>Cross-Sensor Processing Environment</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DMSP</td>
<td>Defense Meteorological Satellite Program</td>
</tr>
<tr>
<td>DOS</td>
<td>Disk Operating System</td>
</tr>
<tr>
<td>DPE</td>
<td>Data Processing Engine</td>
</tr>
<tr>
<td>DPEAS</td>
<td>Data Processing and Error Analysis System</td>
</tr>
<tr>
<td>EOS</td>
<td>Earth Observing System</td>
</tr>
<tr>
<td>F90</td>
<td>Fortran 90</td>
</tr>
<tr>
<td>GCTPC</td>
<td>General Cartographic Transformation Package C</td>
</tr>
<tr>
<td>GIF</td>
<td>Graphics Interchange Format</td>
</tr>
<tr>
<td>GOES</td>
<td>Geostationary Operational Environmental Satellites</td>
</tr>
<tr>
<td>HDF</td>
<td>Hierarchical Data Format</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Photographic Experts Group</td>
</tr>
<tr>
<td>MB</td>
<td>Megabyte</td>
</tr>
<tr>
<td>McIDAS</td>
<td>Man computer Interactive Data Access System</td>
</tr>
<tr>
<td>MFHDF</td>
<td></td>
</tr>
<tr>
<td>MS</td>
<td>Microsoft</td>
</tr>
<tr>
<td>NOAA</td>
<td>National Oceanic and Atmospheric Administration</td>
</tr>
<tr>
<td>OLS</td>
<td>Operational Linescan System</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>RAMDAS</td>
<td>Regional Atmospheric Model Data Assimilation System</td>
</tr>
<tr>
<td>SSMIS</td>
<td>Special Sensor Microwave Imager/Sounder</td>
</tr>
<tr>
<td>TIFF</td>
<td>Tagged Image File Format</td>
</tr>
<tr>
<td>TMI</td>
<td>TRMM Microwave Imager</td>
</tr>
<tr>
<td>TRMM</td>
<td>Tropical Rainfall Measuring Mission</td>
</tr>
<tr>
<td>UNC</td>
<td>Universal Naming Convention</td>
</tr>
<tr>
<td>VIRS</td>
<td>Visible and Infrared Scanner</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>XDR</td>
<td>External Data Representation Standard</td>
</tr>
<tr>
<td>ZLIB</td>
<td></td>
</tr>
</tbody>
</table>
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 [Note: The DPEAS parallelism features are unavailable on Linux and AIX platforms].

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 (F95). All operations are accomplished through subroutine or function calls. Thus, the operational data processing is easy to monitor and change.

The design and development of DPEAS is detailed in Jones et al. (2002), which appears in Appendix A. The remainder of this document describes the modification and use of DPEAS, including augmenting DPEAS scripting language and creating new DPEAS modules.
2 OVERVIEW OF THIS DOCUMENT

This DPEAS Programmer’s Guide, tells Fortran programmers how to add new capabilities to DPEAS and how to modify existing capabilities. A companion document, the DPEAS User’s Guide details how to install and use the current capabilities of DPEAS. 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.

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

Section 3 of this document covers the installation and configuration of DPEAS for programmers. Section 4 covers programming within DPEAS. Section 5 covers the DPEAS program modules and details their interdependencies. Section 6 - Appendix A is a reference to the main DPEAS journal article that appeared in JTECH in 2002. Section 7 - Appendix B is a comprehensive list of DPEAS modules, excluding modules that are in development. Section 8 – Appendix C contains a DPEAS tutorial, using the Jones hdfeos2latlon module as an example.

2.1 DPEAS Versions

Any sections of text that are highlighted in yellow remain a work in progress.

DPEAS version 3.x denotes the fully ported version of DPEAS to Windows/Linux/AIX environments. The OSPO version of DPEAS is called DPEAS_X within the OSPO IBM AIX OPS environment. Differences between OS versions is documented separately in the modules.xls Excel spreadsheet. Current module updates, and technical notes are also listed in the modules.xls spreadsheet.
3 INSTALLATION AND CONFIGURATION

This section is intended to facilitate the smooth installation and configuration of DPEAS. Several topics are covered: (1) DPEAS system requirements, (2) installing DPEAS, (3) configuring DPEAS, (4) DPEAS Compiler Settings, and (5) Updating the DPEAS Compiler Tool Settings.

3.1 DPEAS System Requirements

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

- Windows 2003 (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 recompile DPEAS, your computer must have the following available:

- Microsoft Visual Studio 2008 C++
- Intel Visual Fortran 11.1
- Approximately 850 MB of disk space for the DPEAS project workspace

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, 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

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. To verify the existence of the BJS application, go to Start | Control Panel | Add or Remove Programs and see if the Batch Job Server software has a valid entry. If so, it is installed. If not, it is not installed. 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”

To examine the membership of local user groups enter the following commands at a DOS command prompt: `net localgroup "Batch Users"` and `net localgroup "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 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:

```plaintext
&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, that machine is moved 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 under the DPE_REQUIREMENTS routine description in the DPEAS User’s Guide).

RESOURCE%MEMORY specifies the maximum available memory (in megabytes). DPEAS has the ability to skip resources that do not meet pending job block memory (more information is available under the DPE_REQUIREMENTS routine description in the DPEAS User’s Guide).

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, please 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.

3.4 DPEAS Compiler Settings

DPEAS is a compilable Windows console application. All source code is provided. The configuration for the build is contained within the MS Visual Studio development environment. It is beyond the scope of this document to discuss all features of this program. See MS Visual Studio Help for more information.

Due to system resource requirements, it is suggested that any project modifications are built incrementally. To perform a full rebuild, delete all the files under the .\Debug and/or .\Release\ directories. This will delete the original object files and allow a clean rebuild. Rebuilding the DPEAS library should not be necessary.

The following notes may be useful in compiling DPEAS procedures. It is important to not change the settings below when compiling DPEAS or unexpected compilation errors may occur.

Updating the DPEAS Visual Studio project settings is discussed in section 3.5.
3.4.1 Known Compiler Bugs and Possible Recompilation Issues

There are no known compiler bugs in the current DPEAS build. However, some versions of Windows may have the Visual Numerics (VNI) IMSL libraries in different installation directories, those directory names in the link command may need to be modified to recompile DPEAS on those systems. The Jones-specific path names may also require modification on other systems.

3.4.2 Current Compiler Settings

DPEAS currently runs under the compilers listed below

- Intel Visual Fortran 11.1
- MS Visual Studio 2008 C++

Note: the old compiler and linker options are provided below for comparison purposes. Old options are a partial listing of the main option differences. A full listing of the old compiler and linker options is contained in the following subsections. The DPEAS text file “compiler_settings.txt” will have the most up to date information on the compiler settings. When in doubt, see the actual Visual Studio solution file configuration settings.

3.4.2.1 Current Debug Options


Additional Old C compiler options: /G6 /Gm /GX /ZI /D " _CONSOLE" /FR"Debug/" /Fp"Debug/DPEAS.pch" /YX /GZ

Current Linker options: /OUT:"Debug\PEAS.exe" /INCREMENTAL:NO /NOLOGO /LIBPATH:"..\Debug" /LIBPATH:"C:\Program Files (x86)\VNI\ims\fn1600\IA32\lib" /MANIFEST /MANIFESTFILE:"C:\Users\Jones\Desktop\USB\V1\Projects\DEEP_V1\Components\Windows\C1DOE_V1\DPEAS\src\DPEAS\DPEAS\Debug\DPEAS.exe.intermediate.manifest" /DEBUG /PDB:"C:\Users\Jones\Desktop\USB\V1\Projects\DEEP_V1\Components\Windows\C1DOE_V1\DPEAS\src\DPEAS\DPEAS\Debug\DPEAS.pdb" /SUBSYSTEM:CONSOLE /STACK:6000000
/IMPLIB:"C:\Users\Jones\Desktop\USB\V1\Projects\DEEP_V1\Components\Windows\C1DOE_V1\DPEAS\src\DPEAS\DPEAS\Debug\DPEAS.lib" user32.lib  
Advapi32.lib pdh.lib Clib.lib

Additional Old Linker options: kernel32.lib gdi32.lib winspool.lib comdlg32.lib  
shell32.lib ole2.lib oleaut3.lib uuid.lib odbc32.lib odbccp32.lib  
/subsystem:console /profile /debug /machine:I386

Post-build step options: description: "Incrementing Build Number..."

Command: sh .\setup\configuration\build\build.ksh

3.4.2.2 Current Release Options

Current Fortran compiler options: /nologo /module:"Release" /object:"Release" /libs:static  
/threads /c /Qopenmp-link:static

Additional Old Fortran compiler options: /assume:byterecl /automatic /fpe:0  
/include:"Library/Release/" /noaltparam /optimize:5 /recursive  
/warn:nofileopt /warn:unused

"-_MBCS" /FD /EHsc /MT /Fo"Release" /Fd"Release\vc90.pdb" /W3 /nologo /c /TP  
/wd4996 /errorReport:prompt

Additional Old C compiler options: /ML /GX /D "-_CONSOLE" /Fp"Release/DPEAS.pch"  
/YX /FD

Current Linker options: /OUT:"Release\DPEAS.exe" /INCREMENTAL:NO /NOLINKO  
/LIBPATH:".\Release" /LIBPATH:"C:\Program Files (x86)\VNI\ims1\fn1600\IA32\lib" /MANIFEST  
/MANIFESTFILE:"C:\Users\Jones\Desktop\USB\V1\Projects\DEEP_V1\Components\Windows\C1DOE_V1\DPEAS\src\DPEAS\DPEAS\Release\DPEAS.exe.intermediate.manifest" /SUBSYSTEM:CONSOLE /STACK:6000000  
/IMPLIB:"C:\Users\Jones\Desktop\USB\V1\Projects\DEEP_V1\Components\Windows\C1DOE_V1\DPEAS\src\DPEAS\DPEAS\Release\DPEAS.lib" user32.lib  
Advapi32.lib pdh.lib Clib.lib /LTCG

Additional Old Linker options: kernel32.lib gdi32.lib winspool.lib comdlg32.lib  
shell32.lib ole2.lib oleaut3.lib uuid.lib odbc32.lib odbccp32.lib  
pdb:"Release/DPEAS.pdb" /machine:I386

3.4.2.3 Current Library Options

Listed by module or file specific settings:

Current HDF_41: Project Settings, and /W2 replaces /W3, and /I  
"Source\clib\EOS\JPEG_61" /I "Source\clib\EOS\ZLIB", and /D "INTEL86"
Current MFHDF_41: Project Settings, and /W2 replaces /W3, and /I "Source\clib\EOS\HDF_41"/I "Source\clib\EOS\XDR", and /D "SWAP"/D "NO_SYS_XDR_INC"/D "INTEL86"/D "HDF"

Current GCTPC_C13: Project Settings, and /W2 replaces /W3

Current JPEG: Project Settings, and /W2 replaces /W3

Current XDR: Project Settings, and /W2 replaces /W3

Current ZLIB: Project Settings, and /W2 replaces /W3

Current HDFEOS_25: Project Settings, and /W2 replaces /W3, and /I "Source\clib\EOS\HDF_41"/I "Source\clib\EOS\MFHDF_41", and /D "INTEL86"/D "HDF"

Current CiraErrDLL.mc Description: Performing Custom Build Step on $(InputPath)

Current CiraErrDLL.mc Commands: mc -s -v -h $(InputDir) -r $(InputDir) 
   $(InputDir)\$(InputName).mc

Current CiraErrDLL.mc Outputs:
   $(InputDir)\$(InputName).rc
   $(InputDir)\$(InputName).h

Current McIDAS: Project Settings, and /Zp1 replaces /Zp8, and /I "Source\clib\CiraErrUtil"

Current ols_read.c: Project Settings, and /W2 replaces /W3, and /I "Source\clib\EOS\xdr", and /D "NO_SYS_XDR_INC"

Current goes_read.cpp: Project Settings, and /I "source\lib\McIDAS"/I "Source\clib\CiraErrUtil"

Current avhrr_read.cpp: Project Settings, and /I "source\lib\McIDAS"/I "Source\clib\CiraErrUtil"

Current tmi_read.c: Project Settings, and /I "Source\clib\EOS\MFHDF_41"/I "Source\clib\EOS\HDF_41", and /D "INTEL86"/D "HDF"

Current virs_read.c: Project Settings, and /I "Source\clib\EOS\MFHDF_41"/I "Source\clib\EOS\HDF_41", and /D "INTEL86"/D "HDF"

3.4.3 Older Compiler Settings (for reference)

The older DPEAS compiler settings were for the compilers listed below:

- Compaq Visual Fortran 6.6B
- MS Visual C++ 6.0 SP6
3.4.3.1 Debug Options

**Fortran compiler options:** /architecture:pn3 /assume:byterecl /automatic
   /browser:"Debug/" /check:bounds /check:overflow /compile_only /dbglibs
   /debug:full /fpe:0 /include:"Library/Debug/" /list:"Debug/" /noaltparam
   /nologo /recursive /show:include /stand:f95 /threads /traceback
   /warn:unused /module:"Debug/" /object:"Debug/"
   /pdbfile:"Debug/DF60.PDB"

**C compiler options:** /nologo /G6 /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG"
   /D "_MBCS" /D "_CONSOLE" /FR"Debug/" /Fp"Debug/DPEAS.pch" /YX
   /Fo"Debug/" /Fd"Debug/" /FD /GZ /c

**Linker options:** pdh.lib kernel32.lib user32.lib gdi32.lib winspool.lib
   comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib
   odbc32.lib odbccp32.lib /nologo /stack:0x10000000 /subsystem:console
   /profile /debug /machine:I386 /out:"Debug/DPEAS.exe"

3.4.3.2 Release Options

**Fortran compiler options:** /architecture:pn3 /assume:byterecl /automatic
   /compile_only /fpe:0 /include:"Library/Release/" /noaltparam /nologo
   /optimize:5 /recursive /stand:f95 /threads /tune:pn3
   /module:"Release/" /object:"Release/"

**C compiler options:** /nologo /ML /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_CONSOLE"
   /D "_MBCS" /Fp"Release/DPEAS.pch" /YX /Fo"Release/" /Fd"Release/" /FD /c

**Linker options:** pdh.lib kernel32.lib user32.lib gdi32.lib winspool.lib
   comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib
   odbc32.lib odbccp32.lib /nologo /stack:0x10000000 /subsystem:console
   /incremental:no /pdb:"Release/DPEAS.pdb" /machine:I386 /out:"Release/DPEAS.exe"

3.4.3.3 Library Options

Listed by module or file specific settings:

**HDF_41:** Project Settings, and /W2 replaces /W3, and /I "Source\clib\EOS\JPEG_61"/I
   "Source\clib\EOS\ZLIB", and /D "INTEL86"

**MFHDF_41:** Project Settings, and /W2 replaces /W3, and /I
   "Source\clib\EOS\HDF_41"/I "Source\clib\EOS\XDR", and /D "SWAP"/D
   "NO_SYS_XDR_INC"/D "INTEL86"/D "HDF"

**GCTPC_C13:** Project Settings, and /W2 replaces /W3

**JPEG:** Project Settings, and /W2 replaces /W3
XDR: Project Settings, and /W2 replaces /W3

ZLIB: Project Settings, and /W2 replaces /W3

HDFEOS_25: Project Settings, and /W2 replaces /W3, and /I "Source\clib\EOS\HDF_41"/I "Source\clib\EOS\MFHDF_41", and /D "INTEL86"/D "HDF"

CiraErrDLL.mc Description: Performing Custom Build Step on $(InputPath)

CiraErrDLL.mc Commands: mc -s -v -h $(InputDir) -r $(InputDir) $(InputDir)\$(InputName).mc

CiraErrDLL.mc Outputs:
$(InputDir)\$(InputName).rc
$(InputDir)\$(InputName).h

McIDAS: Project Settings, and /Zp1 replaces /Zp8, and /I "Source\clib\CiraErrUtil"

ols_read.c: Project Settings, and /W2 replaces /W3, and /I "Source\clib\EOS\xdr", and /D "NO_SYS_XDR_INC"

goes_read.cpp: Project Settings, and /I "source\lib\McIDAS"/I "Source\clib\CiraErrUtil"

avhrr_read.cpp: Project Settings, and /I "source\lib\McIDAS"/I "Source\clib\CiraErrUtil"

tmi_read.c: Project Settings, and /I "Source\clib\EOS\MFHDF_41"/I "Source\clib\EOS\HDF_41", and /D "INTEL86"/D "HDF"

virs_read.c: Project Settings, and /I "Source\clib\EOS\MFHDF_41"/I "Source\clib\EOS\HDF_41", and /D "INTEL86"/D "HDF"

3.5 Updating the DPEAS Compiler Tool Settings

To update the DPEAS Visual Studio Project Settings, find “Project Setting” from the menu and follow the instructions below.

Normally the Win32 debug version is used for testing purposes, and then an optimized Win32 release version is created. Verify that the executable directory location and the program arguments list is appropriate and rebuild as necessary.

Step 1: For each configuration “Win32 Debug” and “Win32 Release”, update the executable directory location.

Step 2: For each configuration, update the program argument to point to your input script file location.

Step 3: Rebuild as necessary.
4 PROGRAMMING DPEAS (TBR)

DPEAS is a single monolithic executable. The key to modifying DPEAS is to understand DPEAS’ self-replicating nature. DPEAS has three “phases” (see Figure 1). The first is the compilation phase, which incorporates new program components within the system. This is done with a program hook, a wrapper routine, and the application routine. The second phase begins when the DPEAS executable, DPEAS.exe, is invoked by the OS. The DPEAS program reads from a DPEAS input script to determine what actions it should perform. Normally DPEAS is run on some scheduled basis, where it processes all newly received data sets. As the DPEAS executable interprets the DPEAS input script, it automatically performs parallelization of the input. This is the third phase of DPEAS. During this phase, DPEAS self-replicates to other available nodes. To do this task, DPEAS dynamically emits more DPEAS input script code and evokes multiple copies of itself to run on remote systems. All remote invocations of DPEAS refer to the parent DPEAS executable. Thus, DPEAS distributes itself across the network in a controlled fashion. If no parallelization code is found in original DPEAS input script, phase 3 is not performed. Phase 2 is always performed.

Figure 1: The Relation of the “User_Module.f90” to DPEAS Input Scripts
4.1 MS Visual Studio Primer

4.1.1 Running in Debug Mode with the MS Visual Studio Environment
1. Select “Win32 Debug” in the active configuration selector dialog box
2. Set a breakpoint in the source code using the Build toolbar’s “hand” (F9)
3. Start execution in debug mode (Debug | Start Debugging), then enter “Go” (F5) from the Build toolbar
4. Wait until the program reaches one of your breakpoints
5. Examine variables, set watches, etc.
6. Stop the Debugger or wait for the program to exit on its own

4.1.2 Running in Release Mode with the MS Visual Studio Environment
1. Select “Win32 Release” in the active configuration selector dialog box
2. Start execution in release mode (Debug | Start without Debugging)
3. The program’s command line window will remain open after the program exits
4. Close the command line window when finished

4.2 Batch Job Server Primer
Batch Job Server acts as the job execution and coordination agent. The BJS Client is the primary tool for interacting with the agent. See the BJS help system for more information.

4.3 Modifying DPEAS
- All user routines should interface through the module: \DPEAS\src\DPEAS\Apps\User\dpe_user_module.f90
- Each DPEAS user routine added requires a wrapper to interface correctly to the DPEAS interpreter
- All virtual I/O data interfaces are through the DPEAS libraries
- The principle DPEAS library statements are:
  - Generic function found
  - Generic subroutine allocate_hdfeos
  - Generic overloaded operator =
4.3.1 The 3 Programming Steps to Add a User Routine to DPEAS

1. **Insert a program “hook”**
   The program hook makes the main DPEAS program aware of the existence of your wrapper routine.

2. **Create a wrapper routine**
   The wrapper routine tells the DPEAS Fortran interpreter how to parse and interact with your application subroutine arguments.

3. **Create an application routine**
   The application routine performs the “real” work. There are no programming constraints within the application routine.

4.3.2 Things to Remember

1. Start with simple-small-quick examples
   - Simple/Idealized cases
   - Minimal number of files
   - Small array sizes

2. Disable parallel execution
   - Add the statement `call_dpe_slave` to your DPEAS input script file to turn off DPEAS parallelism

3. Use the debugger - It’s easy and it can show you what your code is really doing
   - Use breakpoints to stop inside your own code
   - Verify that your code segment was entered
   - Verify that your code segment was exited

Additional useful DPEAS statements

- Use `call_dpe_write_data_structure` to list the contents of the virtual I/O data structure
- Use `call_dpe_write_variables` to list the contents of the DPE Fortran interpreter variables
- Use `dpe_code_check` to verify that the fortran source style conforms to the DPEAS standard.
4.3.3 Examples

The following example is a quick overview of what a new DPEAS routine looks like when integrated into the DPEAS software. The source code of the example is at \DPEAS\src\DPEAS\Apps\User\dpe_user_module.f90.

The first stage is to create a user program hook (Figure 2). This connects (or “hooks”) the new code to the existing DPEAS software.

This then calls the user’s “wrapper routine” (Figure 3). The wrapper routine performs all the DPEAS input argument parsing and syntax checking for the user’s new code.

dpe_read_argument, dpe_write_argument, and dpe_call are key routines used by the wrapper routine. They interact with the DPEAS interpreter.

The subroutine example_emissivity calls the user’s application code. This code is now completely isolated within DPEAS system. The user’s application routine (Figures 4-8) is straightforward. The new code declarations declare appropriate file name variables to contain the arguments (Figure 4), and it also declares data pointers to hold the data (Figure 5).

The virtual I/O is performed via Fortran 90 array pointers (Figure 6) using the “found” function. This is a heavily overloaded function and can be used for all supported data types. The allocate_hdfeos routine is used to create memory space in the DPEAS virtual memory system (Figure 7) and in practice using the pointers is very simple (Figure 8).

Usage of the new code within a DPEAS input script is quite simple (Figure 9) and the results are easily verifiable if the new code writes status update messages as it processes the data (Figure 10).

The example shown (Figures 11 and 12) creates a new HDF-EOS output file containing a simple microwave emissivity product. By using a write_hdfeos and a write_gif statement in the DPEAS input script, the output can be written as an HDF-EOS and a GIF file with no new code on the part of the user.

Figure 2: The User’s Program Hook
Figure 3: The User’s Wrapper Routine

```
subroutine call_example_emissivity
  ! Wrapper subroutine for example_emissivity.
  !
  ! modules
  ! use dpe_argument_module
  ! use system_module
  !
  ! DPE subroutine arguments
  ! character (len = MAX_PATH) :: ir_input_file   ! IR input file name
  ! character (len = MAX_PATH) :: mw_input_file   ! MW input file name
  ! character (len = MAX_PATH) :: output_file     ! output file name
  !
  ! local variables
  ! logical :: status     ! logical return status
  !
  ! executable statements
  !
  ! read input arguments
  status = dpe_read_argument (ir_input_file, 'ir_input_file', INTENT = 'IN')
  status = dpe_read_argument (mw_input_file, 'mw_input_file', INTENT = 'IN')
  status = dpe_read_argument (output_file, 'output_file', INTENT = 'IN')
  if (dpe_call()) call example_emissivity (ir_input_file, mw_input_file, output_file)
  !
  return
  !
end subroutine call_example_emissivity
```

4 lines of executable code
Figure 4: The User’s Application Routine Declarations (1 of 2)

```fortran
subroutine example_emissivity (ir_input_file, mw_input_file, output_file)

! Example emissivity subroutine
!
modules
use dpe_data_module
use error_module
!
arguments
character (len = '*'), intent (IN) :: ir_input_file ! IR input file name
character (len = '*'), intent (IN) :: mw_input_file ! MW input file name
character (len = '*'), intent (IN) :: output_file ! output file name
!
local parameters
character (len = '*'), parameter :: PROCEDURE_NAME = 'EXAMPLE_EMISISSIVITY'
real, parameter :: MISSING_VALUE = -1.0 ! missing data value
!
integer :: ir_fillvalue ! dummy integer
real :: ir_fillvalue ! IR fill value
real :: mw_fillvalue ! MW fill value
!
type (type_dpe_data_file), pointer :: file_ir ! IR file DS
!
type (type_dpe_data_file), pointer :: file_mw ! MW file DS
!
type (type_dpe_data_file), pointer :: file_output ! output file DS
!
type (type_dpe_data_grid), pointer :: grid_ir ! IR grid DS
!
type (type_dpe_data_grid), pointer :: grid_mw ! MW grid DS
!
type (type_dpe_data_grid), pointer :: grid_output ! output grid DS
!
type (type_dpe_data_item), pointer :: data_ir ! IR data item DS
!
type (type_dpe_data_item), pointer :: data_mw ! MW data item DS
!
integer :: MAX_HDFE09_STR ! data item names
real, pointer :: emissivity (:, :) ! IR temperatures
real, pointer :: ir (:, :) ! IR temperatures
real, pointer :: mw (:, :) ! MW temperatures
!
! non-executable statements
!
data name '/TB_Ch4_01', 'TB_Ch4_02', 'TB_Ch4_03', 'TB_Ch4_04', 'TB_Ch4_05' /
```

Figure 5: The User’s Application Routine Declarations (2 of 2)

- Pointers to the virtual I/O data structures
- Define arrays as pointers
Figure 6: The User’s Application Routine Using Existing Virtual I/O Data Structures

One function is used to find ALL virtual I/O data structure pointers:

```c
if (.not. found (ir, dpe_data, ir_input_file)) &
call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', 'ir_input_file') &
call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', 'IMAGER') &
call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', 'TB_IR_04')
if (.not. found (data_ir, grid_ir%data, 'TB_IR_04')) &
call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', 'TB_IR_04') &
call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', 'TB_IR_04') &
call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', 'TB_IR_04')
if (.not. found (ir_fillvalue, data_ir)) ir_fillvalue = MISSING_VALUE
```

Figure 7: The User’s Application Routine Creating New Virtual I/O Data Structures

One subroutine is used to allocate ALL virtual I/O data structures:

```c
if (.not. found (file_mw, dpe_data, mw_input_file)) &
call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', 'mw_input_file') &
call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', 'AMSUB') &
call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', 'AMSUB')
```
Figure 8: The User’s Application Routine Using the Virtual I/O Data via Pointers

1. Find each MW channel
2. Allocate a new output array data structure

Your science code looks like this:

```
1. Find each MW channel
2. Allocate a new output array data structure
```

Figure 9: Usage of the New User Routine in a DPEAS Input Script File
Figure 10: The Results: Complete Integration

The new user routine is now fully integrated into DPEAS

Figure 11: The output HDF-EOS file
Another example of integrating a routine into the DPEAS software can be found in Appendix C.
5 DPEAS MODULES

A complete electronic version of the DPEAS module flowchart is available in DPEAS 3.x Flowchart.vsd.

5.1 Creating a New DPEAS Application Routine

This section describes the process of modifying the DPEAS user module file, dpe_user_module.f90. This is the easiest way to modify the DPEAS source code to add new capabilities. Creating a brand new root-level user module is described in section 5.2.

5.1.1 Example

The routine EXAMPLE_EMISSIVITY will be added in the following example. Follow the steps of section 4.3.1, The 3 Programming Steps to Add a User Routine to DPEAS.

1. The root-level program hook for DPEAS user modules already exists in the dpe_syntax_module.f90. A secondary function called dpe_call_user_subroutine provides for a local program hook for EXAMPLE_EMISSIVITY within a case statement structure. This merely calls the appropriate wrapper routine, call_example_emissivity.

2. The wrapper routine, call_example_emissivity, performs the necessary script-level argument parsing. This is done through the dpe_read_argument routines. If output arguments are to be used, the dpe_write_argument routine could be used in a similar manner. The program hook and wrapper routine are called twice by DPEAS. The first call performs a simple syntax check. The second calls (indicated by dpe_call returning .TRUE.) invokes the actual application routine, in this case, EXAMPLE_EMISSIVITY.

3. The application routine, EXAMPLE_EMISSIVITY, performs the “real” work. It is typical that DPEAS application routines use the DPEAS data module, dpe_data_module, and DPEAS error module, error_module, heavily. As discussed in section 4.3, several DPEAS library routines are used frequently (Example: found, allocate_hdfeos, and an overloaded = operator). Fortran pointers are used liberally. It is recommended that users code by example to better learn about the various DPEAS library functionalities. Each callable DPEAS routine has in-code documentation. It is beyond the scope of this document to give a description of each routine. Appendix B contains a module-by-module breakout of the various DPEAS routines. For a better understanding, it is recommended that users start by exploring the code using the electronic version of the DPEAS module flowchart, DPEAS 3.x Flowchart.vsd, as a guide.
5.2 Creating a New Root-Level Module

This section is intended to facilitate the smooth addition of a new root-level DPEAS user module. Carefully follow the steps enumerated below.

Note: In the examples that accompany each step, the user “Smith” was added by modifying the code of “Forsythe,” an existing user module. This process does not impede the full functionality and independence of the user module “Forsythe.”

1. Open a solution workspace by double-clicking on \DPEAS\src\DPEAS\DPEAS.sln.

2. Create a new folder for the new user under the \DPEAS\src\DPEAS\Apps directory [Example: Smith].

3. Copy the files \DPEAS\src\DPEAS\Apps\Forsythe\forsythe_module.f90 and moe_driver_module.f90 into the new user directory.

4. Rename forsythe_module.f90 with the new user’s name [Example: smith_module.f90].

5. Rename moe_driver_module.f90 with the new user’s name [Example: smith_driver_module.f90].

6. In smith_driver_module.f90 replace all instances of moe_ with the new user’s name [Example: _smith].
   
   Note: After completing this step, you should no longer find forsythe or moe in smith_driver_module.f90.

7. In dpe_applications_module.f90, search for use forsythe_module. Add the new module below use forsythe_module [Example: use smith_module].

8. In dpe_call_forsythe, add the new user’s name.
   [Example:
   if (dpe_call_smith_subroutine(subroutine_name)) return].

9. Check that the Developer Studio configuration is using the debugger.

10. Highlight “Build” and select “rebuild all.”

11. Under the directory called “scripts,” write a script file [Example: call_smith_driver].

12. Execute DPEAS with the new user script as the argument. You should add a print statement such as “Hello I’m in Smith Module” in smith_driver_module and see if it appears.
6 Appendix A: “A Dynamic Parallel Data-Computing Environment for Cross-Sensor Satellite Data Merger and Scientific Analysis”
7 APPENDIX B: DPEAS MODULES

The following is a list of DPEAS modules and their exposed public routines. The routines below are organized by their uppercase module group names.

The DPEAS Lib1 library routines…

7.1 COMMON: constants_module

Constant definition module.

7.2 COMMON: dpeas_module

Common DPEAS module that contains all commonly used DPEAS Lib1 modules.

7.3 COMMON: io_module

I/O unit module.

Summary Information:

Module Functions:
- connect_status
- copyfile
- create_filename
- create_filename_ramdas
- get_unit

Module Subroutines:
- filename
- get_files
- get_files_filter
- swap_bytes
- swap_word

Generic Module Functions:
- rename

7.4 COMMON: math_module

Math module.
Summary Information:

Module Functions:
- arc

Generic Module Functions:
- dtr
- rtd

Generic Module Subroutines:
- add_gaussian_noise
- add_uniform_noise
- sort
- zenith_angle

7.5 COMMON: physics_module

Physics module.

Summary Information:

Module Functions:
- rh2q
- satvap_ice
- satvap_water

7.6 COMMON: range_module

Range definition module.

Summary Information:

Generic Module Subroutines:
- area_init
- range_init

7.7 COMMON: sector_module

Sector definition module.

Summary Information:

Module Functions:
• sector_area

Module Subroutines:
• sector_init

Generic Module Functions:
• within_area

7.8 COMMON: time_module

Time module.

Summary Information:

Module Functions:
• leap_year
• time_hdfes_time_to_real_time
• time_real_time_to_hdfes_time
• time_yy_to_yyyy
• time_yyyy_to_yy
• time_now
• time_increment

Generic Module Functions:
• abs
• char
• float
• huge
• max
• min
• time_error

Assignment (=) Interfaces:
• mm = time_string_month
• time_string_month = mm
• hmsc = secs
• secs = hmsc
• yyyyddd = yyyyymmdd
• yyyyymmdd = yyyyddd
• yyyyddd = yyyyddsecs
• secs = yyyyddsecs
• secs = kind_time_real
• time = kind_time_real
• kind_time_real = time
• time = time_bcd
• time = yyyyddsecs
• yyyydddsecs = time
• time_string = string
• time_string = time
• secs = time_string
• yyyyddd = time_string
• yyyydddsecs = time_string
• time = time_string
• time_string_delta = string
• time_string_delta = time
• time = time_string_delta
• time_string_weekday = time

Operator (+) Interfaces:
• yyyydddsecs = yyyyddd + secs
• yyyydddsecs = secs + yyyyddd
• time = time + time
• time = time + kind_time_real
• time = kind_time_real + time

Operator (-) Interfaces:
• time = time - time
• time = time - kind_time_real
• time = kind_time_real - time
• time = -time

Operator (/) Interfaces:
• time = time / time
• time = time / kind_time_real
• time = kind_time_real / time

Operator (*) Interfaces:
• time = time * time
• time = time * kind_time_real
• time = kind_time_real * time

Operator (<) Interfaces:
• kind_time_real < time
• time < kind_time_real
• time < time
• secs < secs

Operator (<=) Interfaces:
• kind_time_real <= time
• time <= kind_time_real
• time <= time
• secs <= secs
Operator (==) Interfaces:
- \texttt{kind\_time\_real \ == \ time}
- \texttt{time \ == \ kind\_time\_real}
- \texttt{time \ == \ time}
- \texttt{secs \ == \ secs}

Operator (>) Interfaces:
- \texttt{kind\_time\_real \ > \ time}
- \texttt{time \ > \ kind\_time\_real}
- \texttt{time \ > \ time}
- \texttt{secs \ > \ secs}

Operator (\geq) Interfaces:
- \texttt{kind\_time\_real \ \geq \ \ time}
- \texttt{time \ \geq \ \ kind\_time\_real}
- \texttt{time \ \geq \ \ time}
- \texttt{secs \ \geq \ \ secs}

7.9 DATA: \texttt{dpe\_data\_allocate\_module}

DPE data allocation module.

Summary Information:

Generic Module Functions:
- \texttt{allocate\_hdfeos}

Module Subroutines:
- \texttt{allocate\_hdfeos\_scan\_position}

7.10 DATA: \texttt{dpe\_data\_assignment\_i1\_module}

DPE data assignment module for integer (\texttt{kind = KIND\_I1}) variables.

Summary Information:

Module Subroutines:
- \texttt{dpe\_data\_element\_assignment\_i1}

7.11 DATA: \texttt{dpe\_data\_assignment\_i2\_module}

DPE data assignment module for integer (\texttt{kind = KIND\_I2}) variables.
Summary Information:

Module Subroutines:
- dpe_data_element_assignment_i2

7.12 DATA: dpe_data_assignment_i4_module
DPE data assignment module for integer variables.

Summary Information:

Module Subroutines:
- dpe_data_element_assignment_i4

7.13 DATA: dpe_data_assignment_module
DPE data assignment module.

Summary Information:

Module Subroutines:
- dpe_data_element_assignment

Assignment (=) Interfaces:
- out_data_info = in_data_info
- out_data_item_list = in_data_item_list
- out_dim_list = in_dim_list
- out_swath = in_swath
- out_var_list = in_var_list

7.14 DATA: dpe_data_assignment_r4_module
DPE data assignment module for real variables.

Summary Information:

Module Subroutines:
- dpe_data_element_assignment_r4

7.15 DATA: dpe_data_assignment_r8_module
DPE data assignment module for double precision variables.
Summary Information:

Module Subroutines:
- dpe_data_element_assignment_r8

7.16 DATA: dpe_data_copy_module
General DPE data copy module.

Summary Information:

Module Subroutines:
- dpe_copy_data

7.17 DATA: dpe_data_create_module
General DPE create module.

Summary Information:

Module Subroutines:
- dpe_create_data

7.18 DATA: dpe_data_deallocate_module
DPE data deallocate module.

Summary Information:

Module Subroutines:
- call_deallocate_hdfeos

Generic Module Functions:
- deallocate_hdfeos

7.19 DATA: dpe_data_find_module
DPE data find module.

Summary Information:

Module Subroutines:
• find_data

7.20 DATA: dpe_data_found_module

DPE data found module.

Summary Information:

Generic Module Functions:
• found

7.21 DATA: dpe_data_hdfeos_gdread_module

DPE data HDF-EOS grid read module.

Summary Information:

Module Subroutines:
• dpe_read_hdfeos_grid

7.22 DATA: dpe_data_hdfeos_gdwrite_module

DPE data HDF-EOS grid write module.

Summary Information:

Module Subroutines:
• dpe_write_hdfeos_grid_list

7.23 DATA: dpe_data_hdfeos_ptread_module

DPE data HDF-EOS point read module.

Summary Information:

Module Subroutines:
• dpe_read_hdfeos_point

7.24 DATA: dpe_data_hdfeos_ptwrite_module

DPE data HDF-EOS point write module.
Summary Information:

Module Subroutines:

- dpe_write_hdfeos_point_list

7.25 DATA: dpe_data_hdfeos_read_module

DPE data HDF-EOS read module.

Summary Information:

Module Subroutines:

- call_read_hdfeos
- read_hdfeos

7.26 DATA: dpe_data_hdfeos_swread_module

DPE data HDF-EOS swath read module.

Summary Information:

Module Subroutines:

- dpe_read_hdfeos_swath

7.27 DATA: dpe_data_hdfeos_swwrite_module

DPE data HDF-EOS swath write module.

Summary Information:

Module Subroutines:

- dpe_write_hdfeos_swath_list

7.28 DATA: dpe_data_hdfeos_utility_module

DPE data HDF-EOS utility module.

Summary Information:

Module Functions:

- dpe_attr_count
- dpe_dim_string_c
Module Subroutines:
  • dpe_define_info

7.29 DATA: dpe_data_hdfeos_write_module

DPE data HDF-EOS write module.

Summary Information:
Module Subroutines:
  • call_write_hdfeos
  • write_hdfeos

7.30 DATA: dpe_data_module

DPE data module.

Note: This is the highest level DPE data module, and as such it should be the only DPE data module explicitly "used" in any user written modules.

Summary Information:
Generic Module Functions:
  • dpe_rename

7.31 DATA: dpe_data_structure_module

DPE data structure module.

Note: This module is the key DPE data structure module.

Summary Information:
Module Functions:
  • dpe_attribute

7.32 DATA: dpe_data_utility_module

DPE data utility module.
Summary Information:

Module Subroutines:
- call_dpe_write_data_structure
- dpe_init_data_info_multisensor
- dpe_init_data_projection
- dpe_read_data_projection_file
- dpe_write_data_structure

Generic Module Functions:
- dpe_data_eq
- dpe_data_ne
- dpe_data_size

Generic Module Subroutines:
- dpe_data_numbertype_error
- dpe_data_rank_error

7.33 DPE: dpe_argument_module

Summary Information:

Module Functions:
- dpe_call

Generic Module Functions:
- dpe_read_argument
- dpe_write_argument

7.34 DPE: dpe_common_module

DPE common module.

Summary Information:

Module Functions:
- dpe_whitespace_line

Module Subroutines:
- dpe_whitespace_input_line

Generic Module Subroutines:
- dpe_errmsg_syntax
7.35 DPE: dpe_database_module

Summary Information:

Module Functions:
- dpe_find_node_database
- dpe_read_slave_output_file

Module Subroutines:
- call_dpe_write_db_statistics
- dpe_accumulate_job_counters
- dpe_close_database
- dpe_open_database
- dpe_slave
- dpe_update_resource_file
- dpe_update_system_bytes
- dpe_write_byte_count
- dpe_write_slave_output_file

Generic Module Subroutines:
- dpe_update_cache_bytes
- dpe_update_total_bytes_read
- dpe_update_total_bytes_written

Generic Assignment Interface:
- type_dpe_byte_count = integer
- type_dpe_byte_count = double_precision

7.36 DPE: dpe_utility_module

DPE utility module.

Summary Information:

Module Functions:
- dpe_copyfile

Module Subroutines:
- call_archive
- call_copyfile
- call_delete_null_files
- call_filename
- call_get_files
- call_system
- call_time
- call_www_update
- dpe_time

7.37  DPE: dpe_variable_module

DPE variable module.

7.38  ERRORS: error_message (facility, level, identification, message)

Generic error message routine whose behavior is dependent on various "on_error" state flags contained within the error module. The five error levels described in Table 1 are recognized.

TABLE 1: RECOGNIZABLE ERROR LEVELS

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘S’</td>
<td>Success</td>
<td>if (verbose) prints success message</td>
</tr>
<tr>
<td>‘I’</td>
<td>Informational</td>
<td>if (verbose) prints informational message</td>
</tr>
<tr>
<td>‘W’</td>
<td>Warning</td>
<td>if (verbose) prints warning message</td>
</tr>
<tr>
<td>‘R’</td>
<td>Return</td>
<td>if (verbose) prints warning message and returns from the current subtask job</td>
</tr>
<tr>
<td>‘E’</td>
<td>Error</td>
<td>if (verbose) prints error message and conditionally stops the system</td>
</tr>
<tr>
<td>‘F’</td>
<td>Fatal</td>
<td>if (verbose) prints fatal message and stops the system</td>
</tr>
</tbody>
</table>

7.39  ERRORS: error_module

DPEAS error module.

Notes: The main error_message subroutine is an external subroutine.

Summary Information:

Module Subroutines:
- errmsg_reset_error_count
- errmsg_set_error_max

Generic Module Subroutine:
- errmsg
7.40  FUSION: dpe_composite_method_module
DPE composite method module. Define all composite methods and related suffixes.

Summary Information:

   Module Function:
   • composite_suffix

7.41  FUSION: dpe_composite_module
DPE composite module.

Summary Information:

   Module Subroutines:
   • call_composite
   • call_composite_weights

7.42  FUSION: dpe_composite_output_module
DPE composite output module.

Summary Information:

   Module Subroutines:
   • define_output_file

7.43  FUSION: dpe_composite_overlay_module
DPE composite overlay module.

Summary Information:

   Module Subroutines:
   • dpe_composite_overlay

7.44  FUSION: dpe_composite_uniform_module
DPE composite uniform module.

Summary Information:
Module Subroutines:
  • dpe_composite_uniform

7.45  FUSION: dpe_composite_w_module

DPE composite weights module.

Summary Information:

Module Subroutines:
  • call_composite_weights

7.46  FUSION: dpe_composite_w_output_module

DPE composite weights output module.

Summary Information:

Module Subroutines:
  • define_output_file

7.47  FUSION: dpe_projection_module

DPE projection module.

Summary Information:

Module Subroutines:
  • deg2dms
  • dms2deg

Generic Module Subroutines:
  • proj_ll
  • proj_ll2xy
  • proj_xy
  • proj_xy2ll

7.48  FUSION: dpe_remap_data_field_module

Remap data field module.

Summary Information:
Module Subroutines:
  • remap_data_field

7.49 FUSION: dpe_remap_data_item_module

DPE remap data item module.

Summary Information:

Module Subroutines:
  • remap_data_item

7.50 FUSION: dpe_remap_module

DPE remap module.

Summary Information:

Module Subroutines
  • call_remap

7.51 FUSION: dpe_remap_output_module

DPE remap output module.

Summary Information:

Module Subroutines:
  • define_output_data
  • define_output_file

7.52 HDFEOS: hdfeos_grid_module

HDF-EOS grid module.

7.53 HDFEOS: hdfeos_module

HDF-EOS module.
7.54 HDFEOS: hdfeos_point_module

HDF-EOS point module.

7.55 HDFEOS: hdfeos_swath_module

HDF-EOS swath module.

7.56 LISTS: list_module

Double-linked list module.

**Summary Information:**

Module Functions:

- add_to_after
- add_to_before
- add_to_head
- add_to_tail
- initialize
- finalize
- find
- size
- write

7.57 LISTS: list_sub_module

Double-linked list sub module.

**Summary Information:**

Module Functions:

- add_to_after
- add_to_before
- add_to_head
- add_to_tail
- initialize
- finalize
- find
- size
- write
7.58 LISTS: string_module

String module.

Summary Information:

Module Functions:
- index_quote
- scan_quote
- str_quote

Module Subroutines:
- str_compress
- str_element
- str_get_quote
- str_lowercase
- str_memory_check
- str_random
- str_uppercase

Generic Module Functions:
- char
- finalize
- len
- read
- str_read
- str_write
- write

Generic Module Subroutines:
- str_add_commas
- str_allocate
- str_append
- str_deallocate

Assignment (=) Interfaces:
- character = string
- string = character
- string = string
- string array = character
- string array = string array

Binary Operator (==) Interfaces:
- character == string
- string == character
- string == string
7.59 OUTPUT: dpe_gif_module

DPE GIF module.

Summary Information:

Module Functions:
• write_output_gif

7.60 OUTPUT: dpe_output_module

DPE output module.

Summary Information:

Module Subroutines:
• call_write_gif
• call_write_ramdas
• call_write_text
• call_write_tiff

7.61 OUTPUT: dpe_ramdas_header_module

DPE RAMDAS header module.

Summary Information:

Module Subroutines:
• write_output_ramdas_header
• write_output_ramdas_line

7.62 OUTPUT: dpe_ramdas_module

DPE RAMDAS module.

Summary Information:

Module Subroutines:
• write_output_ramdas

7.63 OUTPUT: dpe_text_field_module

DPE text field module.
Summary Information:

Module Subroutines:
• write_output_text_field

7.64 OUTPUT: dpe_text_header_module

DPE text header module.

Summary Information:

Module Subroutines:
• write_output_text_header
• write_output_text_line

7.65 OUTPUT: dpe_text_module

DPE text module.

Summary Information:

Module Subroutines:
• write_output_text

7.66 OUTPUT: dpe_tiff_module

DPE TIFF module.

Summary Information:

Module Functions:
• write_output_tiff

7.67 SYSTEM: clib_interface_module

C library interface module.

Summary Information:

Module Functions:
• avhrr_read
• copyfile
Module Subroutines:
  • gctp
  • sys_peak_memory

7.68 SYSTEM: kind_module
Initialize the kind parameters.

7.69 SYSTEM: system_constants_module
System constant definition module.

7.70 SYSTEM: system_module
System dependent module.

Notes: In general, all of the system dependent routines contained in this module can be recreated for any target OS. The major challenge to migrating DPEAS to another OS is obtaining the various compatible C/C++ libraries.

In designing DPEAS, all system dependent Fortran code has been localized to this module, the CLIB interface module (clib_interface_module.f90), the Kind module (kind_module.f90), the System Constants module (system_constants_module.f90) and the HDFEOS modules (hdfeos_grid_module.f90, hdfeos_point_module.f90, and hdfeos_swath_module.f90) with minor exceptions (dpe_test_load_module.f90 and mdias_module.f90) An up-to-date list of exceptions can be found by searching for DEC$NOSTRICT within the Fortran source code.

The practical implications of this is that DPEAS can move to another OS system if 1) an HDF-EOS library is available for the target OS, 2) the various C libraries needed for the particular translators to be supported are available, and 3) if this system module is updated for the target OS. The most likely challenge with this system module will be finding similar file lock mechanisms on the target OS. Some systems may not have as sophisticated file locking methods and may need to have the locks implemented with temporary files, which act as the locking semaphores. Semaphores are an integral part of the Windows OS. The interfaces from the Fortran to C libraries should also be investigated for any substantial
differences. It is my experience that most compiler bugs will be found in those interface specifications. With time, the Fortran and C standards should evolve to support a standardized interface.

Summary Information:

Module Functions:

- sys_cd
- sys_copyfile
- sys_drive_bytes
- sys_file_delete
- sys_file_dir
- sys_file_dir_check
- sys_file_exits
- sys_file_find
- sys_file_find_info
- sys_file_find_new_context
- sys_file_info
- sys_file_name
- sys_file_node
- sys_file_purge
- sys_file_root
- sys_file_size
- sys_file_type
- sys_get_env
- sys_get_tempdir
- sys_get_unit
- sys_makedir
- sys_open
- sys_rename
- sys_system
- sys_time_cpu
- sys_time_elapsed

Module Subroutines:

- sys_close
- sys_create_mirror
- sys_deallocate_mirror
- sys_exit
- sys_get_arg
- sys_get_total_bytes_read
- sys_get_total_bytes_written
- sys_mirror_name
- sys_mirror_synchronize
- sys_mirror_target
- sys_sleep
- sys_write_mirror_sets
7.71 TRANSLATORS: agrmet2hdfeos_module

AGRMET HDF-EOS data translator module.

Summary information:

Module Subroutines:
- call_agrmet2hdfeos

7.72 TRANSLATORS: amsu2hdfeos_module

NOAA AMSU HDF-EOS data translator module.

Summary Information:

Module Subroutines:
- call_amsu2hdfeos

7.73 TRANSLATORS: avhrr2hdfeos_module

AVHRR HDF-EOS data translator module.

Summary Information:

Module Subroutines:
- call_avhrr2hdfeos

7.74 TRANSLATORS: dpe_calibration_module

DPE calibration module.

IMPORTANT: Radiances are given in units of \((\text{mW} / [\text{m}^2 \text{ sr cm}^{-1}])\), thus they are wavenumber-based radiances. Please do not mix wavelength-based or frequency-based radiance units with this module. All microwave radiances should be derived from their equivalent brightness temperature values.

For reference, the following Planck function conversions apply:

\[ B_v (T) = c \times B_f (T) = w^2 \times B_w (T) \]

where,

- \( B \) is the Planck function
- \( T \) is temperature
- \( v \) is wavenumber
\[ f \] is frequency
\[ \omega \] is wavelength

Table 2: Calibration Documentation Matrix

<table>
<thead>
<tr>
<th>SATELLITE</th>
<th>SENSOR</th>
<th>COUNT2RAD</th>
<th>RAD2TB</th>
<th>TB2RAD</th>
<th>RAD2COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMSP</td>
<td>OLS</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMSP</td>
<td>SSM/I</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMSP</td>
<td>SSM/T-2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOES</td>
<td>IMAGER</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>NOAA</td>
<td>AMSU-A</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOAA</td>
<td>AMSU-B</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOAA</td>
<td>AVHRR</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-</td>
</tr>
<tr>
<td>TRMM</td>
<td>TMI</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRMM</td>
<td>VIRS</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note:
- Denotes that the default conversion is used (1 to 1)
- X Denotes channel number type routines
- * Denotes calibration array type routines
- - Denotes calibration array type routines which have not yet been rigorously tested

Summary Information:

Module Functions:
- frequency_dmsp_ssmi
- frequency_dmsp_ssmt2
- frequency_noaa_amsua
- frequency_noaa_amsub
- frequency_trmm_tmi
- wavelength_dmsp_ols
- wavelength_trmm_virs

Module Subroutines:
- info2satnumber
- planck_f
- planck_n
- planck_w
- tb_f
- tb_n
- tb_w
- satnumber2info
Generic Module Functions:
- count2rad
- count2tb
- rad2count
- rad2tb
- tb2count
- tb2rad

7.75 TRANSLATORS: dpe_translator_module
DPE HDF-EOS data translator modules.

7.76 TRANSLATORS: gdas2hdfeos_module
GDAS GRIB data translator module.
Summary Information:
- Module Subroutines:
  - call_gdas2hdfeos

7.77 TRANSLATORS: gfs2hdfeos_module
GFS GRIB data translator module.
Summary Information:
- Module Subroutines:
  - call_gfs2hdfeos

7.78 TRANSLATORS: goes2hdfeos_module
GOES HDF-EOS data translator module.
Summary Information:
- Module Subroutines:
  - call Goes2hdfeos

7.79 TRANSLATORS: mcidas_module
McIDAS data structure module.
7.80 TRANSLATORS: mirs_img2hdfeos_module

MIRS IMG HDF-EOS data translator module.

Summary Information:

Module Subroutines:

- call_mirs_img2hdfeos

7.81 TRANSLATORS: mirs_img_netcdf2hdfeos_module

MIRS IMG netCDF4 HDF-EOS data translator module.

Summary Information:

Module Subroutines:

- call_mirs_img_netcdf2hdfeos

7.82 TRANSLATORS: ols2hdfeos_module

DMSP OLS HDF-EOS data translator module.

Summary Information:

Module Subroutines:

- call_ols2hdfeos

7.83 TRANSLATORS: qmorph2hdfeos_module

QMORPH data translator module.

Summary Information:

Module Subroutines:

- call_qmorph2hdfeos

7.84 TRANSLATORS: ssmi2hdfeos_module

DMSP SSM/I HDF-EOS data translator module.

Summary Information:

Module Subroutines:
• call_ssmi2hdfeos

7.85 TRANSLATORS: ssmis2hdfeos_module
DMSP SSMIS HDF-EOS data translator module.
Summary Information:
    Module Subroutines:
    • call_ssmis2hdfeos

7.86 TRANSLATORS: ssmis_edr2hdfeos_module
DMSP SSMIS EDR HDF-EOS data translator module.
Summary Information:
    Module Subroutines:
    • call_ssmis_edr2hdfeos

7.87 TRANSLATORS: ssmt22hdfeos_module
DMSP SSM/T-2 HDF-EOS data translator module.
Summary Information:
    Module subroutines:
    • call_ssmt22hdfeos

7.88 TRANSLATORS: tmi2hdfeos_module
TRMM TMI HDF-EOS data translator module.
Summary Information:
    Module Subroutines:
    • call_tmi2hdfeos

7.89 TRANSLATORS: virs2hdfeos_module
TRMM VIRS HDF-EOS data translator module.
Summary Information:

Module Subroutines:
- call_virs2hdfeos

7.90 TRANSLATORS: windsat2hdfeos_module

WindSat HDF-EOS data translator module.

Summary Information:

Module Subroutines:
- call_windsat2hdfeos

The DPEAS NESDIS library routines… these routines are independent of the Lib1 library.

7.91 NESDIS: landem_v2_module

NESDIS land emissivity (version 2) module.

Summary Information:

Module Subroutines:
- Landem2

7.92 NESDIS: landem_v3_module

NESDIS land emissivity (version 3) module.

Summary Information:

Module Subroutines:
- Landem3

7.93 NESDIS: NESDIS_OCEANEM_Module

NESDIS ocean emissivity module.

Summary Information:

Module Subroutines:
7.94 NESDIS: Type_Kinds

NESDIS type kinds module.

7.95 NESDIS: weng_emiss_model

NESDIS land emissivity (version 2) module.

Summary Information:

Module Subroutines:

- Landem2

The DPEAS OSPO library routines... these routines are dependent on the Lib1 libraries. The routines are organized by author. The OSPO version of DPEAS (DPEAS_X) employs a blended_tpw_module.f90 that aggregates DPEAS application modules as necessary from the OSPO library source codes. Please see the source codes for more details.

7.96 OSPO/FORSYTHE: btpw_land_blend_module

Blended TPW land module (version 1).

Summary Information:

Module Subroutines:

- btpw_land_blend

7.97 OSPO/FORSYTHE: btpw_land_blend_v2_module

Blended TPW land blend module (version 2).

Summary Information:

Module Subroutines:

- btpw_land_blend_v2
7.98 OSPO/FORSYTHE: cimss_goes_sounder_module

CIMSS GOES Sounder module.

Summary Information:

Module Subroutines:

- call_goes_snodr_tpw_merge
- goes_cimss_read_snodr_tpw_ascii
- btpw_read_sndr_ascii

7.99 OSPO/FORSYTHE: compute_tpw_percent_module

Compute TPW percent module.

Summary Information:

Module Subroutines:

- call_compute_tpw_percent

7.100 OSPO/FORSYTHE: forsythe_utility_module

Forsythe utility module.

Summary Information:

Module Subroutines:

- sparse_data_interpolate
- sort_john4
- gps_all_data_interpolate

Module Functions:

- num_uniq_elements_i4

7.101 OSPO/FORSYTHE: gps_tpw_interpolate_module

GPS TPW interpolate module.

Summary Information:

Module Subroutines:

- call_gps_tpw_interpolate
- gps_tpw_interpolate
• gps_read_fsl_ascii

7.102 OSPO/JONES: ccda_module

The CCDA module (used to read the NVAP data sets).

Summary Information:

Modules Subroutines:

• ccda_read

7.103 OSPO/KIDDER: binomial_smother_module

The Kidder binomial smoother module.

Summary Information:

Module Subroutines:

• call_bsmooth

7.104 OSPO/KIDDER: sqk_mcidas_module

The Kidder McIDAS module.

Summary Information:

Module Subroutines:

• call_write_mcidas

7.105 OSPO/KIDDER: sqk_pentad_correction_module

The Kidder pentad correction module.

Summary Information:

Module Subroutines:

• call_apply_tpw_correction
• call_create_tpw_correction
7.106 OSPO/KIDDER: sqk_rr_correction_module

The Kidder rainfall rate correction module.

Summary Information:

Module Subroutines:

- call_apply_rr_correction
- call_rr_statistics

The DPEAS APPS library routines... these routines are dependent on the Lib1 libraries. The routines are organized by author. The APPS libraries are source codes under user-level source code control. Only select APPS codes are ready for wider distribution and sharing, please contact the DPEAS users directly if you wish to share codes. Please see the source codes for more details.

7.107 APPS: dpe_applications_module

Applications DPE module.

Summary Information:

Module Subroutines:

- dpe_call_applications_subroutine

7.108 COMBS: combs_module

Combs User DPE module.

Summary Information:

Module Subroutines:

- dpe_call_combs_subroutine

7.109 FORSYTHE: forsythe_module

Forsythe User DPE module.

Summary Information:

Module Subroutines:

- dpe_call_forsythe_subroutine
7.110 KIDDER: kidder_module
Kidder User DPE module.

Summary Information:

Module Subroutines:
- dpe_call_kidder_subroutine

7.111 USER: dpe_user_module
DPE user module.

Summary Information:

Module Subroutines:
- dpe_call_user_subroutine

7.112 USER: example_emissivity_module
Example emissivity module.

Summary Information:

Module Subroutines:
- call_example_emissivity

The DPEAS Jones library routines... these routines are dependent of the Lib1 libraries. Only the routines that are used by the OSDPD are listed. The Jones library OSDPD routines are listed under an OPS folder. The OSDPD version of DPEAS (DPEAS_X) employs a blended_tpw_module.f90 that aggregates DPEAS application modules as necessary from Jones, Kidder, and Forsythe source codes. Please see the OSDPD source codes for more details.

7.113 JONES: jones_module
Jones User DPE module.

Summary Information:

Module Subroutines:
- dpe_call_jones_subroutine
7.114 JONES: ccda_module

CCDA module.

Summary Information:

Module Subroutines:
- ccda_read

The DPEAS Test library routines… these routines are dependent on the Lib1 library.

7.115 TEST: test_call_module

DPE test call module.

Summary Information:

Module Subroutines:
- call_dpe_test_call

7.116 TEST: test_hdfeos_module

DPE test HDF-EOS module.

Summary Information:

Module Subroutines:
- call_dpe_test_hdfeos

7.117 TEST: test_kinds_module

Test the kind parameters.

Summary Information:

Module Subroutines:
- test_kinds

7.118 TEST: test_list_module

Test list driver module.
Summary Information:

Module Subroutines:

- test_lists

7.119 TEST: test_load_module
DPE test load module.

Summary Information:

Module Subroutines:

- call_dpe_test_load

7.120 TEST: test_module
DPEAS test module.

7.121 TEST: test_string_module
Test string driver module.

Summary Information:

Module Subroutines:

- test_strings

The main DPEAS routines… these routines are dependent on the NESDIS, Lib1, Test, Apps, and Jones libraries.

7.122 DPEAS: dpe_input_module
DPE input module.

Summary Information:

Module Subroutines:

- dpe_deallocate_input_data
- dpe_process_input
- dpe_read_input
7.123 DPEAS: dpe_module

Data Processing Engine module.

Summary Information:

Module Subroutines:

- dpe_execute
- dpe_exit
- dpe_init

7.124 DPEAS: dpe_syntax_module

DPE syntax module.

Summary Information:

Module Functions:

- dpe_syntax_executable_program

Module Subroutines:

- dpe_syntax_reset

7.125 DPEAS: dpe_version

DPE version subroutine.

Summary Information:

Subroutine:

- dpe_version

7.126 DPEAS: DPEAS

8 APPENDIX C: DPEAS TUTORIAL

The goal of this tutorial is to demonstrate how a user can write special-purpose compilable Fortran code to interact with the DPEAS software (as opposed to writing a DPEAS input script).

The specific example used here is to write a subroutine that will extract and save the latitudes and longitudes from an HDFEOS file. The subroutine is built up in steps to more clearly show the meanings of the various pieces of code.

Step 1.
The first step is to create a module for the task that contains the skeleton of a subroutine that will do the actual work of extracting the latitudes and longitudes. This module is given below:

```fortran
module hdfeos2latlon_module
! HDFEOS to lat/lon module.
!
! summary information
!
! module subroutines
!   call_hdfeos2latlon
!
! explicitly declare public entities
private
  public :: call_hdfeos2latlon
contains

  subroutine call_hdfeos2latlon
! Wrapper subroutine for hdfeos2latlon.
!
! modules
  use dpe_argument_module
  use system_module

! DPE subroutine arguments
  character (len = MAX_PATH) :: output_dir    ! output directory
  character (len = MAX_PATH) :: output_file   ! output file name
  character (len = MAX_PATH) :: input_file     ! input file name

! local variables
  logical :: status   ! logical return status

! executable statements
!
! read input arguments
  status = dpe_read_argument (input_file, 'input_file', INTENT = 'IN')
  status = dpe_read_argument (output_dir, 'output_dir', INTENT = 'IN')
  status = dpe_read_argument (output_file, 'output_file', INTENT = 'OUT')
  if (dpe_call ()) call hdfeos2latlon (input_file, output_dir, output_file)

! write the output variable
  status = dpe_write_argument (output_file, 'output_file')

  return

end subroutine call_hdfeos2latlon
```
subroutine hdfeos2latlon (input_file, output_dir, output_file)
!

! HDFEOS2LATLON driver subroutine.

! modules
  use error_module
  use io_module

! arguments
  character (len = *), intent (IN) :: input_file          ! input file name
  character (len = *), intent (IN) :: output_dir          ! output directory name
  character (len = *), intent (OUT) :: output_file        ! output file name

! local parameters
  character (len = *), parameter :: PROCEDURE_NAME = 'HDFEOS2LATLON'

! executable statements

! write informational messages
  call errmsg (PROCEDURE_NAME, 'I', 'START', 'started processing')
  call errmsg (PROCEDURE_NAME, 'I', 'INPUT_FILE', input_file)
  call errmsg (PROCEDURE_NAME, 'I', 'OUTPUT_DIR', output_dir)

! create the output file name
  call filename (output_file, input_file, output_dir, SUFFIX_SPEC = '_LATLON')
  call errmsg (PROCEDURE_NAME, 'I', 'OUTPUT_FILE', output_file)

! write informational messages
  call errmsg (PROCEDURE_NAME, 'I', 'END', 'processing complete')

return

end subroutine hdfeos2latlon

end module hdfeos2latlon_module

The module contains two subroutines. The first is a wrapper subroutine called call_hdfeos2latlon. The second is an application subroutine called hdfeos2latlon. It is this latter routine that will (eventually) do the actual work of extracting the geographical coordinates.

The wrapper routine does the following:
- Declares the usage of the needed modules (i.e., it adds the appropriate use statements).
- Declares the needed input arguments (input_file, output_dir, output_file) for the application subroutine.
- Reads the input arguments via the DPEAS intrinsic routine dpe_read_argument
- Calls the application subroutine hdfeos2latlon

The hdfeos2latlon subroutine in its current state only does the following:
- Declares variables and modules
- Writes informational messages (routine entered and exited, error messages)
- Sets up the output file name via the DPEAS intrinsic subroutine filename. The output file will have the same name as the input file but with the string ‘_LATLON’ appended to it.
Step 2.
In this step the top level input file data structure is “found” and assigned to a pointer for future program use. The code shown in boldface is new to this step (only routine `hdfeos2latlon`) is shown:

```fortran
subroutine hdfeos2latlon (input_file, output_dir, output_file)
! HDFEOS2LATLON driver subroutine.
!
! modules
use dpe_data_module
use error_module
use io_module
!
! arguments
character (len = *), intent (IN) :: input_file          ! input file name
character (len = *), intent (IN) :: output_dir         ! output directory name
character (len = *), intent (OUT) :: output_file    ! output file name
!
! local parameters
character (len = *), parameter :: PROCEDURE_NAME = 'HDFEOS2LATLON'
!
! local variables
  type (type_dpe_data_file), pointer :: file_input    ! input file DS
!
! executable statements
!
! write informational messages
  call errmsg (PROCEDURE_NAME, 'I', 'START', 'started processing')
  call errmsg (PROCEDURE_NAME, 'I', 'INPUT_FILE', input_file)
  call errmsg (PROCEDURE_NAME, 'I', 'OUTPUT_DIR', output_dir)
!
! create the output file name
  call filename (output_file, input_file, output_dir, SUFFIX_SPEC = '_LATLON')
  call errmsg (PROCEDURE_NAME, 'I', 'OUTPUT_FILE', output_file)
!
! find the input file data structure
  if (.not. found (file_input, dpe_data, input_file)) &
    call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', input_file)
  call dpe_write_data_structure (file_input)
!
! write informational messages
  call errmsg (PROCEDURE_NAME, 'I', 'END', 'processing complete')
!
return
end subroutine hdfeos2latlon
```

Purpose of the new code:
- **Pointer** `file_input` (of type `type_dpe_data_file`) points to the input file data structure.
- **The use of `dpe_data_module` is needed to access the definition of the `type_dpe_data_file` type.
- The DPEAS intrinsic function `found` returns a pointer (`file_input`) that points to the input data structure. If the `found` function fails, a warning message is printed.
- The DPEAS intrinsic routine `dpe_write_data_structure` writes out the input file data structure.
Step 3.
In this step the top level input file data structure pointer reference is reused to “find” a mid-level grid data structure pointer. This is then used to traverse all grid data structure elements. The code shown in boldface is new to this step:

```fortran
subroutine hdfeos2latlon (input_file, output_dir, output_file)
  ! HDFEOS2LATLON driver subroutine.
  ! modules
  use dpe_data_module
  use error_module
  use io_module
  ! arguments
  character (len = *), intent (IN) :: input_file          ! input file name
  character (len = *), intent (IN) :: output_dir         ! output directory name
  character (len = *), intent (OUT) :: output_file    ! output file name
  ! local parameters
  character (len = *), parameter :: PROCEDURE_NAME = 'HDFEOS2LATLON'
  ! local variables
  type (type_dpe_data_file), pointer :: file_input            ! input file DS
  type (type_dpe_data_grid), pointer :: grid_input    ! input grid DS
  ! executable statements
  ! write informational messages
  call errmsg (PROCEDURE_NAME, 'I', 'START', 'started processing')
  call errmsg (PROCEDURE_NAME, 'I', 'INPUT_FILE', input_file)
  call errmsg (PROCEDURE_NAME, 'I', 'OUTPUT_DIR', output_dir)
  ! create the output file name
  call filename (output_file, input_file, output_dir, SUFFIX_SPEC = '_LATLON')
  call errmsg (PROCEDURE_NAME, 'I', 'OUTPUT_FILE', output_file)
  ! find the input file structure
  if (.not. found (file_input, dpe_data, input_file)) &
     call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', input_file)
  grid_input => file_input%grid%head
  do while (associated (grid_input))
    call errmsg (PROCEDURE_NAME, 'I', 'PROCESSING_GRID', grid_input%name)
    call dpe_write_data_structure (grid_input)
    grid_input => grid_input%next
  enddo
  ! write informational messages
  call errmsg (PROCEDURE_NAME, 'I', 'END', 'processing complete')
  return
end subroutine hdfeos2latlon
```

Purpose of the new code:
- Pointer `grid_input` points to the input grid data structure.
- A loop has been set up to traverse the `file_input` grid, which is a linked list. The routine `errmsg` writes out the name of the element of the list, and routine `dpe_write_data_structure` writes out the element’s contents.
Step 4.
This step allocates new memory space for the output file within the DPEAS data structure list. The code shown in boldface is new to this step (some older code has been omitted for brevity):

```fortran
subroutine hdfeos2latlon (input_file, output_dir, output_file)
! HDFEOS2LATLON driver subroutine.

... (some code omitted)

! local variables
  type (type_dpe_data_file), pointer :: file_input          ! input file DS
  type (type_dpe_data_file), pointer :: file_output   ! output file DS
  type (type_dpe_data_grid), pointer :: grid_input        ! input grid DS

! executable statements

... (some code omitted)

! find the input file data structure
  if (.not. found (file_input, dpe_data, input_file)) &
    call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', input_file)

! allocate the output file data structure
  call allocate_hdfeos (file_output, output_file)
  call dpe_write_data_structure (file_output)

! loop through all available grids
  grid_input => file_input%grid%head
  do while (associated (grid_input))
    call errmsg (PROCEDURE_NAME, 'I', 'PROCESSING_GRID', grid_input%name)
    grid_input => grid_input%next
  enddo

! write informational messages
  call errmsg (PROCEDURE_NAME, 'I', 'END', 'processing complete')

return
end subroutine hdfeos2latlon

Purpose of the new code:
- Pointer file_output points to the output file data structure.
- The routine allocate_hdfeos allocates the file_output variable.
- file_output is written out.
```

Step 5.
This step continues from the previous by allocating new memory space specific to the new data grid of the output file. Each new grid uses the input file grid as a template for the new grid. Each new grid has the same projection information and grid domain size as the corresponding input grid data structures. The code shown in boldface is new to this step:

```fortran
subroutine hdfeos2latlon (input_file, output_dir, output_file)
! HDFEOS2LATLON driver subroutine.
```

```fortran
... (some code omitted)

! allocate the output file data structure
  call allocate_hdfeos (file_output, output_file)
  call dpe_write_data_structure (file_output)

! loop through all available grids
  grid_input => file_input%grid%head
  do while (associated (grid_input))
    call errmsg (PROCEDURE_NAME, 'I', 'PROCESSING_GRID', grid_input%name)
    grid_input => grid_input%next
  enddo

! write informational messages
  call errmsg (PROCEDURE_NAME, 'I', 'END', 'processing complete')

return
end subroutine hdfeos2latlon
```
! modules
use dpe_data_module
use error_module
use io_module
use string_module
use system_module

! arguments
character (len = *), intent (IN) :: input_file          ! input file name
character (len = *), intent (IN) :: output_dir          ! output directory name
character (len = *), intent (OUT) :: output_file       ! output file name

! local parameters
character (len = *), parameter :: PROCEDURE_NAME = 'HDFEOS2LATLON'

! local variables
character (len = MAX_PATH) :: name                     ! dummy character name
type (type_dpe_data_file), pointer :: file_input       ! input file DS
type (type_dpe_data_file), pointer :: file_output      ! output file DS
type (type_dpe_data_grid), pointer :: grid_input       ! input grid DS
type (type_dpe_data_grid), pointer :: grid_output      ! output grid DS

! executable statements
...

! loop through all available grids
grid_input => file_input%grid%head
do while (associated (grid_input))
   call errmsg (PROCEDURE_NAME, 'I', 'PROCESSING_GRID', grid_input%name)

! allocate the output grid data structure
name = grid_input%name
   call allocate_hdfeos (grid_output, file_output%grid, name, grid_input%proj)
   call dpe_write_data_structure (grid_output)

   grid_input => grid_input%next
enddo

! write informational messages
   call errmsg (PROCEDURE_NAME, 'I', 'END', 'processing complete')

return
end subroutine hdfeos2latlon

Purpose of the new code:
- Dummy variable name has been added. The parameter MAX_PATH is declared in system_module.
- Pointer grid_output points to the output grid data structure. It is allocated by routine allocate_hdfeos.
- The string_module is used to allow for assignment between character variables and varying string variables.
- grid_output is written out.
Step 6.
Within this step the output arrays “Latitude” and “Longitude” are created within each newly created output grid data structure. The code shown in boldface is new to this step:

```fortran
subroutine hdfeos2latlon (input_file, output_dir, output_file)
! HDFEOS2LATLON driver subroutine.

! modules
use constants_module
use dpe_data_create_module
use dpe_data_module
use error_module
use io_module
use string_module
use system_module

! local arrays
logical, allocatable :: status (:, :)   ! status array
real, pointer :: lat (:, :)                   ! latitude array
real, pointer :: lon (:, :)                  ! longitude array

! executable statements

! allocate the output grid data structure
name = grid_input%name
call allocate_hdfeos (grid_output, file_output%grid, name, grid_input%proj)

! allocate the lat/lon as part of the new output file
call dpe_create_data (grid_output, 'Latitude', lat, MISSING_LATLON)
call dpe_create_data (grid_output, 'Longitude', lon, MISSING_LATLON)
call dpe_write_data_structure (grid_output)

! allocate a temporary status array
allocate (status (size (lat, DIM = 1), size (lat, DIM = 2)))

! deallocate the temporary status array
deallocate (status)

grid_input => grid_input%next
dendo

! write informational messages
call errmsg (PROCEDURE_NAME, 'I', 'END', 'processing complete')
return
end subroutine hdfeos2latlon
```

Purpose of the new code:
- The pointer arrays `lat` and `lon` have been added for the latitudes and longitudes, and logical array `status` indicates if the geographical coordinates exist. Array `status` is allocated to be the same size as the lat/lon arrays.
- The two calls to routine `dpe_create_data` create data fields for latitude and longitude in `grid_output`. 
• The parameter **MISSING_LATLON** is defined in **constants_module** and is a flag for missing lat/lon values.

**Step 7.**
This step performs the actual computation of the latitude and longitude values using a dpe_projection_module library call **proj_ll**. The code shown in boldface is new to this step:

```fortran
subroutine hdfeos2latlon (input_file, output_dir, output_file)
! HDFEOS2LATLON driver subroutine.
modules
  use constants_module
  use dpe_data_create_module
  use dpe_data_module
  use dpe_projection_module
  use error_module
  use io_module
  use string_module
  use system_module
... (some code omitted)
! allocate a temporary status array
allocate (status (size (lat, DIM = 1), size (lat, DIM = 2)))
! compute the latitude and longitude values
  call proj_ll (grid_output%proj, lat, lon, status)
! deallocate the temporary status array
  deallocate (status)
  grid_input => grid_input%next
enddo
! write informational messages
  call errmsg (PROCEDURE_NAME, 'I', 'END', 'processing complete')
return
end subroutine hdfeos2latlon
```

**Purpose of the new code:**
• The subroutine **proj_ll** (defined in **dpe_projection_module**) is called to do the transformation that produces the latitudes and longitudes.

**Step 8.**
In this step, the DPEAS HDF-EOS information data structure component is initialized, basically “signing” the data file with general information as to the file contents. This is used to keep track of the file contents, and to provide a mechanism for data set version control within the HDF-EOS data format framework. The code shown in boldface is new to this step. The module is shown in its entirety.
module hdfeos2latlon_module
! HDFEOS to lat/lon module.
!
! summary information
!
! module subroutines
! call_hdfeos2latlon
!
! explicitly declare public entities
private
public :: call_hdfeos2latlon
!
contains

subroutine call_hdfeos2latlon
! Wrapper subroutine for hdfeos2latlon.
!
modules
use dpe_argument_module
use system_module
!
DPE subroutine arguments
character (len = MAX_PATH) :: output_dir ! output directory
character (len = MAX_PATH) :: output_file ! output file name
character (len = MAX_PATH) :: input_file ! input file name
!
languages
logical :: status ! logical return status
!
executable statements
!
! read input arguments
status = dpe_read_argument (input_file, 'input_file', INTENT = 'IN')
status = dpe_read_argument (output_dir, 'output_dir', INTENT = 'IN')
status = dpe_read_argument (output_file, 'output_file', INTENT = 'OUT')
if (dpe_call ()) call hdfeos2latlon (input_file, output_dir, output_file)
!
write the output variable
status = dpe_write_argument (output_file, 'output_file')
!
return
!
end subroutine call_hdfeos2latlon
!
PRIVATE ENTITIES · PRIVATE ENTITIES · PRIVATE ENTITIES · PRIVATE ENTITIES

subroutine hdfeos2latlon (input_file, output_dir, output_file)
! HDFEOS2LATLON driver subroutine.
!
modules
use constants_module
use dpe_data_create_module
use dpe_data_module
use dpe_projection_module
use error_module
use io_module
use string_module
use system_module
!
arguments
character (len = *), intent (IN) :: input_file ! input file name
character (len = *), intent (IN) :: output_dir  ! output directory name
character (len = *), intent (OUT) :: output_file ! output file name

! local parameters
character (len = *), parameter :: PROCEDURE_NAME = 'HDFEOS2LATLON'

! local variables
character (len = MAX_PATH) :: name           ! dummy character name

!! type (type_dpe_data_file), pointer :: file_input   ! input file DS
!! type (type_dpe_data_file), pointer :: file_output  ! output file DS
!! type (type_dpe_data_grid), pointer :: grid_input   ! input grid DS
!! type (type_dpe_data_grid), pointer :: grid_output  ! output grid DS

! local arrays
logical, allocatable :: status (:, :) ! status array
real, pointer :: lat (:, :) ! latitude array
real, pointer :: lon (:, :) ! longitude array

! executable statements

! write informational messages
call errmsg (PROCEDURE_NAME, 'I', 'START', 'started processing')
call errmsg (PROCEDURE_NAME, 'I', 'INPUT_FILE', input_file)
call errmsg (PROCEDURE_NAME, 'I', 'OUTPUT_DIR', output_dir)

! create the output file name
call filename (output_file, input_file, output_dir, SUFFIX_SPEC = '_LATLON')
call errmsg (PROCEDURE_NAME, 'I', 'OUTPUT_FILE', output_file)

! find the input file data structure
if (.not. found (file_input, dpe_data, input_file)) &
call errmsg (PROCEDURE_NAME, 'W', 'NOT_FOUND', input_file)

! allocate the output file data structure
  call allocate_hdfeos (file_output, output_file)

! loop through all available grids
  grid_input => file_input%grid%head
  do while (associated (grid_input))
    call errmsg (PROCEDURE_NAME, 'I', 'PROCESSING_GRID', grid_input%name)

! allocate the output grid data structure
    name = grid_input%name
    call allocate_hdfeos (grid_output, file_output%grid, name, grid_input%proj)

! copy the information list from the input grid and update the information for the output grid
  grid_output%info = grid_input%info
  grid_output%info%data_description = 'HDFEOS Latitude and Longitude Data'
  grid_output%info%data_source = 'Derived Product'
  grid_output%info%data_version = 1
  call allocate_hdfeos (grid_output%info, grid_output%attr)

! allocate the lat/lon as part of the new output file
  call dpe_create_data (grid_output, 'Latitude', lat, MISSING_LATLON)
  call dpe_create_data (grid_output, 'Longitude', lon, MISSING_LATLON)

! allocate a temporary status array
  allocate (status (size (lat, DIM = 1), size (lat, DIM = 2)))

! compute the latitude and longitude values
  call proj_ll (grid_output%proj, lat, lon, status)
! deallocate the temporary status array
deallocate (status)

grid_input => grid_input%next
enddo

! write informational messages
call errmsg (PROCEDURE_NAME, 'I', 'END', 'processing complete')
call dpe_write_data_structure (file_output)

return

end subroutine hdfeos2latlon

end module hdfeos2latlon_module

Purpose of the new code:
- The information list is copied from the input to the output grid. The `data_description`, `data_source`, and `data_version` fields are updated for the output grid.
- The `file_output` data structure is written out. This final call to `dpe_write_data_structure` is optional and merely generates a complete file listing of the final file contents.

The subroutine is now complete. When a DPEAS input script is used that calls `hdfeos2latlon`, an output file (`output_file` located in `output_dir`) will be created that stores only the latitudes and longitudes for the specified HDFEOS input file (`input_file`).

9 ACKNOWLEDGEMENTS

The original edition (January 2005) of this DPEAS Programmer’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.