Pablo HDF Trace Utilities
User's Guide

1 Overview

Most HDF applications handle large amounts of data. Handling the data efficiently is important to the performance of the application. The purpose of the ***Pablo HDF Trace Utilities*** is to provide information about the calls to HDF routines and I/O operations, which may be analyzed. This analysis provides a means for tuning code, which uses HDF functions. Visit the HDF website at `http://hdf.ncsa.uiuc.edu` for further information on HDF.

The Pablo HDF Trace Utilities consist of a Pablo instrumented copy of version 4.1R2 of the ***HDF library***, the ***Pablo Trace and Trace Extensions libraries***, and some utilities for processing the output. The instrumented version of the HDF library has hooks inserted into the HDF code which call routines in the Pablo Trace Extension library just after entry to each instrumented HDF routine and just prior to exit from the routine. The Pablo Trace Extension library has routines that track the HDF and I/O activity between the entry and exit of the HDF routine during execution. A few lines of code must be inserted in the users **`main`** program to enable tracing and to specify which HDF procedures are to be traced. The program is linked with the special libraries to produce an instrumented executable. Running this executable produces an output file called the ***trace file*** which contains ***Pablo Self-Defining Data Format*** (***SDDF***) records. These SDDF records can then be processed to extract information about the HDF and I/O activity that occurred during execution. For further information about Pablo Self-Defining Data format, visit the Pablo website at `http://www-pablo.cs.uiuc.edu`.

The Pablo HDF Trace Utilities provide two methods for instrumenting HDF codes, ***Runtime Tracing*** and ***Summary Tracing***.

1.1 Runtime Tracing

If the Runtime Tracing option is used, an SDDF packet called an ***HDF entry trace record*** is written to the trace file each time an HDF procedure is entered. The HDF entry trace record contains information such as the time that the procedure was entered and a code indicating the type of the procedure. On return from a traced HDF procedure, an ***HDF exit trace record*** is produced which contains the time of return. Each I/O operation produces an SDDF record packet containing duration of the operation. At the end of the run, the trace file contains enough information to produce a thorough analysis of the I/O and HDF activity that occurred during execution of the program.

1.2 Summary Tracing

If the Summary Tracing option is used, statistics about the HDF and I/O activity are recorded in tables in memory during runtime. Prior to the end of execution, an SDDF record called an ***HDF***

summary record is produced for each of the HDF procedures that were traced. These records summarize the total number of I/O operations and calls to other traced HDF procedures that occurred within each type of traced HDF procedure during runtime.

The Runtime Tracing option produces much more information than the Summary Tracing but may also produce trace files that are extremely large. The size of the Summary Tracing output file is roughly proportional to the number of HDF procedures being traced.

2 Instrumenting Your Code

Any HDF code that can call ***Fortran*** or ***C*** programs can be instrumented. In order for your code to produce output from HDF and I/O calls, the main program must be modified slightly. The modifications involve inserting a call to initialize tracing before any HDF calls are made and a call to end tracing prior to exit from the program.

2.1 C Interface

If your main program is written in C, add the line

```
HDFinitTrace( traceFile , procMask , traceType ) ;
```

anywhere in the executable portion of the code before the HDF code to be traced and add the line

```
HDFendTrace( );
```

anywhere after the code to be traced. The functions `HDFinitTrace` and `HDFendTrace` are described as follows.

- **HDFinitTrace**

Usage:

```
#include "ProcMasks.h"
void HDFinitTrace ( char *traceFile, unsigned procMask,
                    int traceType );
```

where

`traceFile` is the name of the file where the output is to be written. If Summary Tracing is performed, a file named **`<traceFile>.map`** will also be produced where **`<traceFile>`** is the contents of the character string **`traceFile`**. The `.map` file contains information about the named HDF identifiers found during execution.

`procMask` is a bit string indicating the class of HDF functions to be traced (*see bel*ow)

`traceType` is an integer indicating the type of SDDF output :
`traceType` = 0, no tracing output
`traceType` = 1, perform **Runtime Tracing**
`traceType` = 2, perform **Summary Tracing**

The possible choices of classes, which can be traced, correspond roughly to the capital letters at the beginning of the names of the HDF entry points. For example, to trace such functions as **`SDstart`** and **`SDcreate`**, the ***SD*** class should be traced.

The choices for function classes and the corresponding hexadecimal values of their masks are

DFAN	0x1
DFP	0x2
DFR8	0x4
DFSD	0x8
DF24	0x10
H	0x20
HE	0x40
SD	0x80
VF	0x100
V	0x200
VH	0x400
VS	0x800
AN	0x1000
GR	0x2000
HA	0x4000
DA	0x8000
HUL	0x10000

The include file, **`ProcMasks.h`**, defines **`DFAN_mask`**, **`DFP_mask`**, …, **`HUL_mask`** to have the hexadecimal values specified in the table above. A value of `procMask` equal to `DFAN_mask` would enable tracing of the *DFAN* class of HDF functions, a value of `procMask` equal to `SD_mask` would enable tracing of the *SD* class of HDF functions and so on. The file `ProcMasks.h` lists the names of the routines in the class corresponding to each mask.

The defined masks can be `added` to enable tracing of more than one class at the same time. For example,

```
#include "ProcMasks.h"
      …
procMask = SD_mask + DFP_mask + H_mask ;
HDFinitTrace ( "myFile.out", procMask, 1 );
```

would enable tracing of the *SD*, *DFP* and *H* classes. The output would SDDF packets written to `myFile.out`. The mask **`AllHDF_mask`** has value 0x1FFFF and enables tracing of all HDF

functions. The `ProcMask.h` file is in the **`HDF include`** directory supplied with the HDF package. If **`<hdfDir>`** is the directory containing the HDF programs, then use the flag **`-I<hdfDir>/include`** in the compile step.

- **HDFendTrace:**

Usage:

```
void HDFendTrace();
```

This call causes Pablo cleanup operations to be performed. The output file may be incomplete or nonexistent if this call is not made.

See Example 1 in the Examples section for an illustration of the use of the HDFinitTrace and HDFendTrace functions in a ***C*** program.

2.2 Fortran interface

If your main program is written in Fortran, add the call

```
call HDFinitTrace( traceFile , procMask, traceType )
```

anywhere in the executable portion of the code before the HDF code to be traced and add the call

```
call HDFendTrace( )
```

anywhere after the code to be traced. Because Fortran is not case sensitive, upper or lower case letters can be use for the subroutine names and other constants described below. Mixed case is used here for clarity.

- **HDFinitTrace**

Usage:

```
character*(*) traceFile
include 'ProcMasks.inc'
integer procMask, traceType

call HDFinitTrace ( traceFile, procMask, traceType)
```

where

`traceFile` is the name of the file where the output is to be written. If Summary Tracing is performed, a file named **`<tracefile>.map`** will also be produced where **`<tracefile>`** is the value of the character array **`traceFile`**. The

`.map` file contains information about the named HDF identifiers found during execution.

`procMask` is a bit string indicating the class of HDF functions to be traced (*see bel*ow)

`traceType` is an integer indicating the type of SDDF output :
`traceType` = 0, no tracing output
`traceType` = 1, perform **Runtime Tracing**
`traceType` = 2, perform **Summary Tracing**

The possible choices of classes, which can be traced, correspond to the capital letters at the beginning of the names of the HDF entry points. For example, to trace such functions as **`SDstart`** and **`SDcreate`**, the ***SD*** class should be traced. The choices for function classes and the corresponding hexadecimal values of their masks are:

DFAN	Z'1'
DFP	Z'2'
DFR8	Z'4'
DFSD	Z'8'
DF24	Z'10'
H	Z'20'
HE	Z'40'
SD	Z'80'
VF	Z'100'
V	Z'200'
VH	Z'400'
VS	Z'800'
AN	Z'1000'
GR	Z'2000'
HA	Z'4000'
DA	Z'8000'
HUL	Z'10000'

The include file **`ProcMasks.inc`** defines **`DFANmask`**, **`DFPmask`**, …, **`HULmask`** to have the hexadecimal values specified in the table above. A value of `procMask` equal to `DFANmask` would enable tracing of the *DFAN* class of HDF functions, a value of `procMask` equal to `SDmask` would enable tracing of the *SD* class of HDF functions and so on. The file `ProcMasks.inc` lists the names of the routines in the class corresponding to each mask.

The defined masks can be added to enable tracing of more than one class at the same time. For example,

```
include "ProcMasks.inc"
      ...
procmask = SDmask + DFPmask + Hmask
call HDFinitTrace ( "myfile.out", procmask, 1 )
```

would enable tracing of the ***SD***, ***DFP*** and ***H*** classes. Output would be SDDF packets written to myfile.out. The mask **allhdfmask** has value Z'1FFFF' and enables tracing of all HDF functions. The file **ProcMasks.inc** is in the HDF include directory. If <hdfDir> is the directory containing the HDF programs, then use the flag **-I<hdfDir>/include** in the compile step.

- **HDFendTrace:**

Usage:

```
call HDFendTrace( )
```

This call causes Pablo cleanup operations to be performed. The output file may be incomplete or nonexistent if this call is not made.

See Example 2 in the Examples section for an illustration of the use of the subroutines HDFinintTrace and HDFendTrace in a ***Fortran*** program.

2.3 Linking

The libraries required to enable tracing in your code are the HDF libraries **libmfsd-inst.a** and **libdf-inst.a**, and the Pablo libraries **libPabloTraceExt.a** and **libPabloTrace.a**. In addition, the HDF libraries libz.a and libjpeg.a must be linked in as well. If the HDF libraries are located in the directory **<hdfDir>/lib** and the Pablo Trace Libraries are located in **<pabloDir>/lib**, then the flags

```
-L<hdfDir>/lib -L<pabloDir>/lib -lmfsd-inst -ldf-inst –lz –ljpeg \
-lPabloTraceExt -lPabloTrace
```

should be used in the link step. More libraries may be needed, depending on the application.

3 Processing the Output

The output produced from the execution of the instrumented HDF program is a binary file that must be processed using SDDF utilities. As part of the Pablo HDF Utilities, the executables ***CreateHDFRecords*** and ***HDFStatsTables*** are provided to transform the output file into statistical tables.

3.1 CreateHDFRecords

CreateHDFRecords is used only with Runtime Tracing. This utility must be run on the trace file produced during execution. For each traced HDF procedure call, it summarizes the I/O activity and number and duration of calls to other HDF procedures that occurred. This summary is written to an output file as an SDDF packet.

The syntax for this utility is

CreateHDFRecords tracefile [-o ***outfile***]

where

tracefile is the trace file produced by running the instrumented program, i.e., the value of name specified in the **`traceFile`** parameter for **`HDFinitTrace`** .

If the -o option is selected, the name of the new SDDF file is ***outfile***. If -o is not specified, the output file is ***pabloHDF.bin***.

Execution of CreateHDFRecords also produces another files, one with a ***.map*** suffix. That is, if the –o *outfile* option is used, a files name ***outfile.map*** is produced, and if the –o option is not used, the file ***pabloHDF.bin.map*** is produced. The ***.map*** file contains a list of all the named identifiers of HDF quantities such as files, scientific data sets, and dims that were accessed during execution. For each named identifier, there is a corresponding number assigned to it by Pablo in this ***.map*** file.

3.2 HDFStatsTables

The HDFStatsTables utility can be used either on the output file produced by running CreateHDFRecords on the output from Runtime Tracing or it can be used directly on the output file produced by Summary Tracing. This utility produces statistical tables summarizing the HDF and I/O activity within the program. The user may optionally select to summarize only the activity that is related to a specific named HDF identifier. The tables produced by HDFStatsTables are written to the ***standard output***.

The HDFStatsTables utility is invoked as follows:

HDFStatsTables *[-hdfID* ***i]*** ***hdftracefile***

where

hdftracefile is the file produced by CreateHDFRecords if Runtime Tracing was performed or the trace file specified for *Summary Tracing* output

options:

-hdfID ***i*** specifies that only the HDF and I/O activity associated with the HDF named identifier with associated Pablo Id ***i*** should be reported. The value of ***i*** associated with an HDF named identifier can be obtained from the *.map* file

produced either by CreateHDFRecords if Runtime Tracing was performed or directly from the run if Summary tracing was performed.

3.2.1 Output Tables

The HDFStatsTables utility produces several tables, General HDF Summary Information., the I/O Operation Count Summary, the I/O Duration Summary, and the Bytes Transferred Summary.

3.2.1.1 General HDF Summary Information

This table contains information such as the time of the first and last HDF call, the approximate execution time of the program, the total amount of execution time spent and the percent of execution time spent in HDF procedures and the number of HDF calls. After that there is a summary titled **General HDF Activity** which contains a line for each of the HDF procedures that were traced. The entries in the row are

- the number of calls made to the procedure,
- the inclusive duration time of the procedure, i.e., the total time spent in the procedure,
- the exclusive duration, i.e., the total time spent in the procedure excluding I/O operations and calls to other traced HDF procedures,
- the number of standard I/O operations performed by the procedure,
- the total time spent doing I/O in the procedure,
- the number of MPI-IO operations performed by the procedure,
- the total time spend in MPI-IO operations,
- the number of calls to other traced HDF procedures made by the procedure,
- total time spent in other traced HDF procedures.

An example of the General Summary Information and General HDF Activity Table is given in the section 4.

3.2.1.2 I/O Operation Count Summary

The I/O Operation Count Summary table lists the number of I/O operations performed within each of the traced HDF procedures. It lists only those procedures that performed I/O operations. Each line gives the procedure name and the number of *Read*, *Write*, *Asynchronous Read*, *Asynchronous Write*, *Open*, *Close*, *Seek*, *Wait*, and *Other* I/O operations performed.

An example of an I/O Operation Count Summary Table is given in the section 4.

3.2.1.3 I/O Operation Duration Summary

The I/O Operation Duration Summary table lists the total time spent performing I/O operations performed within each of the traced HDF procedures. It lists only those procedures that performed I/O operations. Each line gives the procedure name and the total time spend performing *Read*,

Write, *Asynchronous Read*, *Asynchronous Write*, *Open*, *Close*, *Seek*, *Wait*, and *Other* I/O operations.

An example of an I/O Operation Duration Summary Table is given in the section 4.

3.2.1.4 Bytes Transferred Summary

The Bytes Transferred Summary table lists the number of bytes transferred by each of the traced procedures. It lists only those procedures that performed I/O operations. Each line gives the number of bytes transferred by *Read*, *Write*, *Asynchronous Read*, and *Asynchronous Write* I/O operations.

An example of a Bytes Transferred Summary Table is given in the section 4.

4 A *C* Language Example

The steps to obtaining the statistical tables of the HDF and I/O activity in your code are

- modify your code to enable instrumentation selecting Runtime or Summary tracing,
- compile your code using the `-I<hdfDir>/include` option, where <`hdfDir`> is the directory containing the HDF software,
- link your object code with the HDF libraries `libmfdf-inst.a`, `libdf-inst.a`, `libz.a`, and `libjpeg.a`, and the Pablo Trace libraries `libPabloTraceExt.a` and `libPabloTrace.a`,
- execute your code to produce an SDDF trace file,
- if Runtime Tracing was used,
  - run CreateHDFRecords on the trace file to produce a second SDDF file,
  - run HDFStatsTables on the second SDDF file
- if Summary tracing was used,
  - run HDFStatsTables on the original SDDF trace file

Next an example is presented to illustrate how to modify, compile, and link a ***C*** language program that will produce a trace file on execution and how to use the CreateHDFRecords and HDFStatsTables utilities to produce statistical tables.

The following is an example of the ***C*** language program prior to modifying it for Pablo HDF tracing. This program is **`gr_ex1.c`** provided with the HDF test suite.

```
#include "hdf.h"
```

```
#include "mfgr.h"

#define X_LENGTH 5
#define Y_LENGTH 10

main( )
{

      int32 gr_id, ri_id, file_id, status;
      int32 dimsizes[2], ncomp, il;

      /* Create and open the file. */
      file_id = Hopen("Example1.hdf", DFACC_CREATE, 0);

      /* Initiate the GR interface. */
      gr_id = GRstart(file_id);

      /* Define the number of components and dimensions of the image. */
      ncomp = 2;
      il = MFGR_INTERLACE_PIXEL;
      dimsizes[0] = X_LENGTH;
      dimsizes[1] = Y_LENGTH;

      /* Create the image array. */
      ri_id = GRcreate(gr_id, "Image_array_1", ncomp, DFNT_INT16, il,
                               dimsizes);

      /* Terminate access to the image array. */
      status = GRendaccess(gr_id);

      /* Terminate access to the GR interface and close the file */
      status = GRend(ri_id);

      /* Close the file. */
      status = Hclose(file_id);
}
```

4.1 Modifying Your Code

Suppose we wish to perform Runtime Tracing of the *GR* and *H* categories of HDF procedures used when this program is executed with the trace output going to the file `gr_ex1.trace`.

The file `ProcMasks.h` should be included and a call to `HDFinitTrace` should be inserted before calls to any HDF procedures. Because the *GR*, *DA* and *H* categories of procedures are to be traced, the value of the `procMask` argument should be `GR_mask + DA_mask + H_mask`. The value of the third argument should be 1 for Runtime Tracing. A call to HDFendTrace must be inserted prior to exiting the program. These modifications produce the following program.

```
#include "hdf.h"
#include "mfgr.h"
#include "ProcMasks.h"  /* define Pablo HDF masks */

#define X_LENGTH 5
#define Y_LENGTH 10

main( )
{

      int32 gr_id, ri_id, file_id, status;
      int32 dimsizes[2], ncomp, il;

      /* Initialize Pablo HDF Tracing */
      HDFinitTrace( "gr_ex1.trace", GR_mask + DA_mask + H_mask , 1 );
      /* Create and open the file. */
      file_id = Hopen("Example1.hdf", DFACC_CREATE, 0);

      /* Initiate the GR interface. */
      gr_id = GRstart(file_id);

      /* Define the number of components and dimensions of the image. */
      ncomp = 2;
      il = MFGR_INTERLACE_PIXEL;
      dimsizes[0] = X_LENGTH;
      dimsizes[1] = Y_LENGTH;

      /* Create the image array. */
      ri_id = GRcreate(gr_id, "Image_array_1", ncomp, DFNT_INT16, il,
                              dimsizes);

      /* Terminate access to the image array. */
      status = GRendaccess(gr_id);

      /* Terminate access to the GR interface and close the file */
      status = GRend(ri_id);

      /* Close the file. */
      status = Hclose(file_id);

      /* End Pablo HDF Tracing */
      HDFendTrace();

}
```

4.2 Compiling Your Code

If the directory where HDF is installed is <`hdfDir`> to compile the program, the command

```
cc –c –O gr_ex1.c -I<hdfDir>/include
```

can be used to produce the object file `gr_ex1.o`.

4.3 Linking Your Code

If the directory where HDF is installed is <hdfDir> and the directory containing the Pablo Trace software is <pabloDir>, the command

```
cc gr_ex1.o –L<hdfDir>/lib –L<pabloDir>/lib -lmfsd-inst \
   -ldf-inst –lz –ljpeg -lPabloTraceExt -lPabloTrace –o gr_ex1
```

can be used to produce an executable named `gr_ex1`. Additional libraries may be required depending on your installation.

4.4 Executing Your Program

The command `gr_ex1` will produce the SDDF trace file `gr_ex1.trace.`

4.5 Obtaining Statistical Tables

First run CreateHDFRecords on `gr_ex1.trace` to produce another SDDF file.

CreateHDFRecords gr_ex1.trace

In this example, the default output file `pabloHDF.bin` is produced. The file `pabloHDF.bin.map` is also produced. The `.map` file contains the following:

```
Pablo ID to HDF Name mappings:
------------------------------
   1  Image_array_1
   2  Example1.hdf
```

Next run HDFStatsTables on the file `pabloHDF.bin` to produce the output tables. These will be written to standard output.

HDFStatsTables pabloHDF.bin

The tables obtained in this case are given below. The timing information will vary depending on the machine.

Note that the command

HDFStatsTables –hdfID 1 `pabloHDF.bin`

would print tables with information restricted to operations involving `Image_array_1`.

General HDF Summary Information

Trace File: pabloHDF.bin

First HDF Function called at 0.04 seconds
Last HDF Function called at 0.10 seconds

Approximate execution time on 1 nodes 0.10 seconds
Approximate wall-clock time 0.10 seconds

Total time spent in HDF procedures 0.06 seconds
Percent execution time spent in HDF procedures 58.62
Number of HDF procedure calls 57

General HDF Activity

This table summarizes the how time is spent in each of the HDF procedures. It contains the following information.

- Inclusive Duration, the total time spent in the procedure.
- Exclusive Duration, the time spent in the procedure excluding IO operations and calls to other traced procedures.
- Total Number of Standard IO operations performed by the procedure and Total Time Spent doing these operations
- Total Number of Calls by the procedure to MPI IO functions and Total Time spent in these functions
- Total Number of Calls by the procedure to other HDF routines and Total Time spent in these routines

Note: All times are measured in seconds

Procedure	N Calls	Inclusive Duration Time	Exclusive Duration Time	Standard IO		MPI IO		Other HDF	
				N OPs	Total Time	N Calls	Total Time	N Calls	Total Time
Hclose	1	0.03	0.00	6	0.03	0	0.00	2	0.00
Hendaccess	2	0.00	0.00	0	0.00	0	0.00	0	0.00
Hfind	8	0.00	0.00	0	0.00	0	0.00	0	0.00
Hgetfilevers’	1	0.00	0.00	0	0.00	0	0.00	0	0.00
Hgetlibversi’	2	0.00	0.00	0	0.00	0	0.00	0	0.00
Hnewref	1	0.00	0.00	0	0.00	0	0.00	0	0.00
Hnumber	5	0.00	0.00	0	0.00	0	0.00	0	0.00
Hopen	1	0.03	0.00	6	0.03	0	0.00	3	0.00
Hputelement	2	0.00	0.00	0	0.00	0	0.00	6	0.00
Hstartread	6	0.00	0.00	0	0.00	0	0.00	6	0.00
Hstartwrite	2	0.00	0.00	0	0.00	0	0.00	4	0.00
Hwrite	2	0.00	0.00	2	0.00	0	0.00	0	0.00
Hstartaccess	8	0.00	0.00	0	0.00	0	0.00	14	0.00
Hsetlength	2	0.00	0.00	0	0.00	0	0.00	0	0.00
HIstart	1	0.00	0.00	0	0.00	0	0.00	0	0.00
HPregister_t’	3	0.00	0.00	0	0.00	0	0.00	0	0.00
GRstart	1	0.00	0.00	0	0.00	0	0.00	14	0.00
GRend	1	0.00	0.00	0	0.00	0	0.00	0	0.00
GRcreate	1	0.00	0.00	0	0.00	0	0.00	2	0.00
GRendaccess	1	0.00	0.00	0	0.00	0	0.00	0	0.00
DAcreate_arr’	2	0.00	0.00	0	0.00	0	0.00	0	0.00
DAdestroy_ar’	2	0.00	0.00	0	0.00	0	0.00	0	0.00
DAset_elem	2	0.00	0.00	0	0.00	0	0.00	0	0.00

’ Indicates procedure name is truncated.

I/O Operation Count Summary

This table lists the number of I/O operations performed within each of the HDF procedures.
Only those procedures performing I/O operations are listed.

Procedure	Read	Write	Asyn Read	Asyn Write	Open	Close	Seek	Wait	Other
Hclose	0	3	0	0	0	1	2	0	0
Hopen	0	3	0	0	1	0	1	0	1
Hwrite	0	2	0	0	0	0	0	0	0
Totals	0	8	0	0	1	1	3	0	1

I/O Operation Duration Summary

This table summarizes the total time in seconds spent performing I/O operations within each of the HDF procedures.
Only those procedures performing I/O operations are listed.

Procedure	Read	Write	Asyn Read	Asyn Write	Open	Close	Seek	Wait	Other
Hclose	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00
Hopen	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00
Hwrite	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Totals	0.00	0.00	0.00	0.00	0.03	0.03	0.00	0.00	0.00

Bytes Transferred Summary

This table summarizes the number of bytes transferred by the HDF procedures.
Only those procedures performing reads or writes are included.

Procedure	Read	Write	Asyn Read	Asyn Write
Hclose	0	199	0	0
Hopen	0	202	0	0
Hwrite	0	107	0	0
Totals	0	508	0	0

5 A Fortran Language Example

The steps for obtaining trace information from a ***Fortran*** program are the same as those used to obtain trace information from a ***C*** program. In this section, only the code modification step is shown.

The following is an example of the ***Fortran*** language program prior to modifying it for Pablo HDF tracing. The program is **`gr_ex1.f`** provided with the HDF test suite.

```
      PROGRAM CREATE IMAGE ARRAY

      integer*4 gr_id, ri_id, file_id, dimsizes(2), ncomp, il
      integer mgstart, mgcreat, mgendac, mgend, hopen, hclose
      integer*4 X_LENGTH, Y_LENGTH, status
      parameter (X_LENGTH = 5, Y_LENGTH = 10,
     +           MFGR_INTERLACE_PIXEL = 0)
C     DFACC_CREATE and DFNT_INT16 are defined in hdf.h.
      integer*4 DFACC_CREATE, DFNT_INT16
      parameter (DFACC_CREATE = 4, DFNT_INT16 = 22)

C     Create and open the file.
      file_id = hopen("Example1.hdf", DFACC_CREATE, 0)

C     Initiate the GR interface.
      gr_id = mgstart(file_id)

C     Define the number of components and dimensions of the image array.
      ncomp = 2
      il = MFGR_INTERLACE_PIXEL
      dimsizes(1) = Y_LENGTH
      dimsizes(2) = X_LENGTH

C     Create the image array.
      ri_id = mgcreat(gr_id, 'Ex_array_1', ncomp, DFNT_INT16, il,
     +                dimsizes)

C     Terminate access to the image array.
      status = mgendac(ri_id)

C     Terminate access to the GR interface and close the file.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)
      end
```

The following is the code after modifying it to do Runtime Tracing of *GR*, *DA* and *H* category of HDF procedures. The trace output file is **`gr_ex1.trace.`**

```
PROGRAM CREATE IMAGE ARRAY

C*    Include HDF mask definitions
      include "ProcMasks.inc"
      integer*4 gr_id, ri_id, file_id, dimsizes(2), ncomp, il
      integer mgstart, mgcreat, mgendac, mgend, hopen, hclose
      integer*4 X_LENGTH, Y_LENGTH, status
      parameter (X_LENGTH = 5, Y_LENGTH = 10,
     +           MFGR_INTERLACE_PIXEL = 0)
C     DFACC_CREATE and DFNT_INT16 are defined in hdf.h.
      integer*4 DFACC_CREATE, DFNT_INT16
      parameter (DFACC_CREATE = 4, DFNT_INT16 = 22)

C*    Enable Pablo HDF tracing of GR, DA and H class functions
      call HDFinitTrace( "gr_ex1.trace", GRmask + Damask + Hmask, 1 )
C     Create and open the file.
      file_id = hopen("Example1.hdf", DFACC_CREATE, 0)

C     Initiate the GR interface.
      gr_id = mgstart(file_id)

C     Define the number of components and dimensions of the image array.
      ncomp = 2
      il = MFGR_INTERLACE_PIXEL
      dimsizes(1) = Y_LENGTH
      dimsizes(2) = X_LENGTH

C     Create the image array.
      ri_id = mgcreat(gr_id, 'Ex_array_1', ncomp, DFNT_INT16, il,
     +                dimsizes)

C     Terminate access to the image array.
      status = mgendac(ri_id)

C     Terminate access to the GR interface and close the file.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)

C*    end Pablo HDF tracing
      call HDFendTrace()
      end
```