Happendable

```
intn Happendable(int32 H_id)
```

H_id	IN:	Access identifier returned by **Hstartwrite**

Purpose	Appends data to a data set without the use of linked blocks.
Return value	Returns `SUCCEED(0)` if data element can be appended and `FAIL(-1)` otherwise
Description	If a data element is at the end of a file **Happendable** allows **Hwrite** to write past the end of the file without the use of linked blocks.
FORTRAN	None

Hclose

hclose

Hclose

```
intn Hclose(int32 file_id)
```

file_id	IN:	The HDF file identifier returned by **Hopen**

Purpose — Closes the access path to the file.

Return value — Returns `SUCCEED(0)` if successful and `FAIL(-1)` otherwise.

Description — The file identifier *file_id* is validated before the file is closed. If the identifier is valid, the function closes the access path to the file.

If there are still access identifiers attached to the file, the error `DFE_OPENAID` is placed on the error stack, `FAIL(-1)` is returned, and the file remains open. This is a fairly common error when developing new interfaces. See **Hendaccess** for discussion of this problem.

FORTRAN

```
integer function hclose(file_id)
    integer*4 file_id
```

Hdeldd

```
intn Hdeldd(int32 file_id, uint16 tag, uint16 ref)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag of data descriptor to delete
ref	IN:	Reference number of data descriptor to delete

Purpose Deletes a tag and reference number from the data descriptor list.

Return value Returns `SUCCEED(0)` if successful and `FAIL(-1)` otherwise.

Description Removes the tag and reference number for a data object from the data descriptor list. Once the data descriptor is removed, the data in the data object becomes inaccessible. To remove inaccessible data from an HDF file, use the utility **hdfpack**.

Hdeldd only deletes the specified tag and reference number from the data descriptor list. Data objects containing the deleted tag and reference number are not automatically updated. For example, if the tag and reference number deleted from the descriptor list referenced an object in a Vgroup, the tag and reference number will still exist in the Vgroup even though the data is inaccessible.

FORTRAN None

Hdupdd

```
intn Hdupdd(int32 file_id, uint16 tag, uint16 ref, uint16 old_tag, uint16 old_ref)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag for the duplicate data descriptor
ref	IN:	Reference number for the duplicate data descriptor
old_tag	IN:	Tag for original data descriptor
old_ref	IN:	Reference number for the original data descriptor

Purpose Generates a new data descriptor for the data in an existing data object.

Return value Returns `SUCCEED(0)` if successful and `FAIL(-1)` otherwise.

Description Duplicates the specified data descriptor in the data descriptor list. Data descriptors contain the following information: `tag`, `reference number`, `length`, and `offset`. When a data descriptor is duplicated, it's `length` and `offset` are added to the data descriptor list under a new tag and reference number.

Because duplicate data descriptors contain the same `length` and `offset` information, both descriptors point to the same data element in the file. In other words, duplicating a data descriptor does not duplicate the data contained in the original data object. It simply creates a second data object by redundantly pointing to the same data location in the file.

The tag and reference number for the duplicate data descriptor are assigned when **Hdupdd** is called. If the specified tag and reference number already exist in the file, the original data descriptor is not duplicated and **Hdupdd** returns an error.

FORTRAN None

Hendaccess

```
intn Hendaccess(int32 H_id)
```

H_id	IN:	Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**

Purpose Terminates access to a data object by disposing of the access identifier.

Return value Returns SUCCEED(0) if successful and FAIL(-1) otherwise.

Description The number of active access identifiers is limited to MAX_ACC as defined in the hdf header file. Because of this restriction, it is very important to call **Hendaccess** immediately following the last operation on a data element.

When developing new interfaces, a fairly common mistake is to omit calling **Hendaccess** for all of the elements accessed. When this happens, **Hclose** will return FAIL, and a dump of the error stack will report how many access identifiers are still active (see **HEprint**).

This is a rather difficult problem to debug, because the low levels of the HDF library cannot determine who and where an access identifier was originated. As a result, there is no automated way to determine which access identifiers have yet to be released.

FORTRAN None

Hfidinquire

```
intn Hfidinquire(int32 file_id, char **filename, intn *access, intn *attach)
```

file_id	IN:	HDF file identifier returned by **Hopen**
filename	OUT:	Complete path and filename for the HDF file
access	OUT:	Access mode file is opened with
attach	OUT:	Number of access identifiers attached to the file

Purpose Inquires about a file by using its file identifier.

Return value Returns `SUCCEED(0)` if successful and `FAIL(-1)` otherwise.

Description Gets the complete path name, access mode, and number of access identifiers associated with a file. Files are identified by passing **Hfidinquire** a *file_id*. The *filename* parameter is a pointer to a character pointer which will be modified when the function returns. Upon completion, *filename* is set to point to the file name in internal storage. All output parameters must be non-`NULL` pointers.

FORTRAN None

Hfind

```
intn Hfind(int32 file_id, uint16 search_tag, uint16 search_ref, uint16 *find_tag,
          uint16 *find_ref, int32 *find_offset, int32 *find_length, intn direction)
```

file_id	IN:	HDF file identifier returned by **Hopen**
search_tag	IN:	The tag to search for or DFTAG_WILDCARD
search_ref	IN:	Reference number to search for or DFREF_WILDCARD
find_tag	IN:	If (**find_tag*==0) and (**find_ref*==0) then start search
	OUT:	Tag matching the *search_tag*
find_ref	IN:	If (**find_tag*==0) and (**find_ref*==0) then start search
	OUT:	Reference number matching the *search_ref*
find_offset	OUT:	Offset of the data element found
find_length	OUT:	Length of the data element found
direction	IN:	Direction to search in - DF_FORWARD searches forward from the current location, and DF_BACKWARD searches backward from the current location

Purpose — Locates the next object of a search in an HDF file

Return value — Returns SUCCEED(0) if successful and FAIL(-1) otherwise.

Description — **Hfind** searches for the next data element that matches the specified tag and reference number. Wildcards apply. If direction is DF_FORWARD, search from current position forwards in the file, otherwise DF_BACKWARD searches backward from the current position in the file.

If *find_tag* and *find_ref* are both set to 0, this indicates the beginning of a search, and the search will start from the beginning of the file if the direction is DF_FORWARD and from the end of the file if the direction is DF_BACKWARD.

FORTRAN — None

Hgetelement

Hgetelement

```
int32 Hgetelement(int32 file_id, uint16 tag, uint16 ref, uint8 *data)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag of the data element to read
ref	IN:	Reference number of the data element to read
data	OUT:	Buffer to read into

Purpose — Obtains the data referred to by the tag and reference number that is passed to it.

Return value — Returns the number of bytes read if successful and `FAIL(-1)` otherwise.

Description — Reads the data element for the specified tag and reference number and puts it into the buffer pointed to by *data*. It is assumed that the space allocated for the buffer is large enough to hold the data.

FORTRAN — None

Hgetfileversion

```
intn Hgetfileversion(int32 file_id, uint32 *major_v, uint32 *minor_v, uint32 *release,
          char string[])
```

file_id	IN:	HDF file identifier returned by **Hopen**
major_v	OUT:	Major version number
minor_v	OUT:	Minor version number
release	OUT:	Release number
string	OUT:	Informational text string

Purpose — Gets version information for an HDF file.

Return value — Returns `SUCCEED(0)` if successful and `FAIL(-1)` otherwise.

Description — Returns the HDF version number stored in the given file. It is still an open question as to what exactly the version number of a file should mean, so we recommend that code not depend on this buffer. The *string* parameter is limited to a length of `LIBVSTR_LEN` characters as defined in `hfile.h`.

FORTRAN — None

Hgetlibversion

Hgetlibversion

```
intn Hgetlibversion(uint32 *major_v, uint32 *minor_v, uint32 *release, char string[])
```

major_v	OUT:	Major version number
minor_v	OUT:	Minor version number
release	OUT:	Release number
string	OUT:	Informational text string

Purpose
Gets version information for current HDF library.

Return value
Returns `SUCCEED(0)` if successful and `FAIL(-1)` otherwise.

Description
Returns the version of the HDF library. The version information is statistically compiled into the HDF library, so it is not necessary to have any open files for this function to execute. The *string* buffer is limited to a length of `LIBVSTR_LEN` characters as defined in `hfile.h`.

FORTRAN
None

Hinquire

```
intn Hinquire(int32 H_id, int32 *file_id, uint16 *tag, uint16 *ref, int32 *length,
          int32 *offset, int32 *position, int16 *access, int16 *special)
```

H_id	IN:	Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**
file_id	OUT:	HDF file identifier returned by **Hopen**
tag	OUT:	Tag of the element pointed to
ref	OUT:	Reference number of the element pointed to
length	OUT:	Length of the element pointed to
offset	OUT:	Offset of the element in the file
position	OUT:	Current position within the data element
access	OUT:	The access type for this data element
special	OUT:	Special code

Purpose — Returns access information about a data element.

Return value — Returns `SUCCEED(0)` if the access identifier points to a valid data element and `FAIL(-1)` otherwise.

Description — Inquires about the statistics of the data object specified by *H_id*. If *H_id* is a valid access identifier the access type (read or write) is set regardless of whether or not the return value is `FAIL (-1)`. If *H_id* is invalid, the function returns `FAIL (-1)` and the access type is set to zero. To avoid excess information, pass `NULL` for any unnecessary pointer.

FORTRAN — None

Hishdf

```
intn Hishdf(char *filename)
```

filename	IN:	Complete path and filename for the file to check

Purpose — Determines if a file is an HDF file.

Return value — Returns `TRUE(1)` if the file is an HDF file and `FALSE(0)` otherwise.

Description — The decision of where a file is and if a file is and HDF file is based solely on the value stored in the first four bytes of an HDF file. It is possible that **`Hishdf`** will identify a file as an HDF file but **`Hopen`** will be unable to open the file (for example if the data descriptor list is corrupt).

FORTRAN — None

HLcreate

```
int32 HLcreate(int32 file_id, uint16 tag, uint16 ref, int32 block_len, int32 n_blocks)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag of new data descriptor
ref	IN:	Reference number of new data descriptor
block_len	IN:	Length of blocks to be used
n_blocks	IN:	Number of blocks to use per linked block record

Purpose

Creates a linked block special data element.

Return value

Returns an access identifier if successful and FAIL(-1) otherwise.

Description

Appending data to existing data objects is a problem in early versions of the HDF library because objects are required to be stored contiguously. When appending, the HDF library deleted the existing element and moved it to the end of the file. HDF 3.2 added the concept of linked block elements which allow unlimited appending without deleting or copying existing data.

HLcreate creates a new linked block element if given an unused tag and reference number. If an existing tag and reference number are used, **HLcreate** promotes the existing data element to a linked block element. To create a linked data element, specify the number of data blocks for the element in *n_blocks* and their length in *block_len*. All data blocks must be the same length unless the element was promoted from a non-linked block data element. In this case, all but the original data block must conform to the length specification.

HLcreate creates linked block elements by inserting a linked block structure between a data descriptor and its associated data element. The linked block structure is used to keep track of the different blocks of data appended to the same data object. All data blocks associated with a new linked block element are stored at the same location. If a linked block element is promoted from a standard data element, the block containing the original data remains in its original position with the remaining data blocks stored together at a different file location. This non-contiguous configuration is known to cause problems with **Hoffset** and **Hlength** which assume that data in a data element is stored contiguously.

Promoting existing data elements to linked block elements will not automatically close opened access identifiers. Open access identifiers prevent a file from closing, therefore it is important to use **Hendaccess** to remove any open identifiers before attempting to close a file.

FORTRAN

None

Hlength

```
int32 Hlength(int32 file_id, uint16 tag, uint16 ref)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag of the data element
ref	IN:	Reference number of the data element

Purpose — Returns the length of a data object specified by the tag and reference number.

Return value — Returns the length of data element if found and `FAIL (-1)` otherwise.

Description — **Hlength** calls **Hstartread**, **Hinquire**, and **Hendaccess** to determine the length of a data element. **Hlength** uses **Hstartread** to obtain an access identifier for the specified data object. If an access identifier already exists (i.e. **Hstartread** was called earlier in the program), it is more efficient to call **Hinquire** directly.

Hlength will return the correct data length for linked block elements, however it is important to remember that the data in linked block elements is not stored contiguously.

FORTRAN — None

Hnewref

```
uint16 Hnewref(int32 file_id)
```

file_id	IN:	HDF file identifier returned by **Hopen**

Purpose	Returns a reference number that is guaranteed to be unique to the file.
Return value	Returns the reference number if successful and 0 otherwise.
Description	Returns a reference number that can be used with any tag to produce a unique tag and reference number pair. Successive calls to **Hnewref** will generate a strictly increasing sequence until the highest possible reference number has been returned, then **Hnewref** will return unused reference numbers starting from 1.
FORTRAN	None

Hnextread

```
intn Hnextread(int32 H_id, uint16 tag, uint16 ref, int origin)
```

H_id	IN:	Access identifier returned by **Hstartread** or previous **Hnextread**
tag	IN:	Tag to search for
ref	IN:	Reference number to search for
origin	IN:	Position to begin search: DF_START or DF_CURRENT

Purpose

Locates and positions a read access identifier on the next data element with the specified tag and reference number.

Return value

Returns SUCCEED(0) if successful and FAIL(-1) otherwise.

Description

Searches for the next data descriptor that matches specified tag and reference number. Wildcards apply. If *origin* is DF_START, the search will start at the beginning of the data descriptor list. If *origin* is DF_CURRENT, the search will begin at the current position. Searching from the end of a file is not yet implemented.

If the search is successful, the access identifier reflects the new data element, otherwise it is not modified.

FORTRAN

None

hnumber

Hnumber

```
int32 Hnumber(int32 file_id, uint16 tag)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag to be counted

Purpose

Returns the number of instances of a tag in a file.

Return value

Returns the number of instances of a tag in a file if successful, and FAIL(-1) otherwise.

Description

Hnumber determines how many objects with the specified tag are in a file. To count the total number of objects in a file, set the *tag* argument to DFTAG_WILDCARD. Note that a return value of zero is not a fail condition.

FORTRAN

```
integer function hnumber(file_id, tag)
    integer*4 file_id
    integer*2 tag
```

Hoffset

```
int32 Hoffset(int32 file_id, uint16 tag, uint16 ref)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag of the data element
ref	IN:	Reference number of the data element

Purpose — Gets the offset for a data element in the file.

Return value — Returns the offset of data element if the data element exists and `FAIL(-1)` otherwise.

Description — **Hoffset** calls **Hstartread**, **Hinquire**, and **Hendaccess** to determine the length of a data element. **Hoffset** uses **Hstartread** to obtain an access identifier for the specified data object. If an access identifier already exists (i.e. **Hstartread** was called earlier in the program), it is more efficient to call **Hinquire** directly.

Hoffset will return the correct offset for a linked block element, however it is important to remember that the data in linked block elements is not stored contiguously. The offset returned by **Hoffset** only reflects the position of the first data block.

Hoffset will also cause problems if used to determine the offset of an external element. In this case, **Hoffset** returns zero, an invalid offset for all HDF files.

FORTRAN — None

Hopen

```
int32 Hopen(char *filename, intn access, int16 n_dds)
```

filename	IN:	Complete path and filename for the file to be opened
access	IN:	DFACC_READ, DFACC_RDWR, or DFACC_CREATE
n_dds	IN:	Number of data descriptors in a block if a new file is to be created

Purpose

Provides an access path to an HDF file by reading all the data descriptor blocks into memory.

Return value

Returns the file identifier if successful and FAIL(-1) otherwise.

Description

If given a new file name, **Hopen** will create a new file using the specified access type and number of data descriptors. If given an existing file name, **Hopen** will open the file using the specified access type and ignore the *n_dds* argument.

HDF provides several constants for use as access codes:

Recommended:

DFACC_READ	Open for read only. If file does not exist, error
DFACC_RDWR	Open for read/write. If file does not exist, create it
DFACC_CREATE	If file exists, delete it, then open a new file for read/write

Others:

DFACC_ALL	Same as DFACC_RDWR
DFACC_WRITE	Same as DFACC_RDWR

If a file is opened and an attempt is made to reopen the file using DFACC_CREATE, HDF will issue the error: DFE_ALROPEN. If the file is opened with read only access and an attempt is made to reopen the file for write access using DFACC_RDWR, DFACC_WRITE, or DFACC_ALL, HDF will attempt to reopen the file with read and write permissions.

Upon successful exit, the named file is opened with the relevant permissions, the data descriptors are set up in memory, and the associated *file_id* is returned. For new files, the appropriate file headers are also set up.

FORTRAN

```
integer function hopen(filename, access, n_dds)
   character*(*)  filename
   integer        access
   integer*2      n_dds
```

Hputelement

```
int32 Hputelement(int32 file_id, uint16 tag, uint16 ref, uint8 *data, int32 length)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag of the data element to add or replace
ref	IN:	Reference number of the data element to add or replace
data	IN:	Pointer to data buffer
length	IN:	Length of data to write

Purpose — Adds or replaces a data element in an HDF file.

Return value — Returns the number of bytes written if successful and `FAIL(-1)` otherwise.

Description — Writes a data element or replaces an existing data element in a HDF file.

FORTRAN — None

Hread

```
int32 Hread(int32 H_id, int32 length, uint8 *data)
```

H_id	IN:	Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**
length	IN:	Length of segment to read in
data	OUT:	Pointer to the data array to read to

Purpose — Reads the next segment in a data element.

Return value — Returns the length of segment actually read if successful and `FAIL(-1)` otherwise.

Description — **Hread** begins reading at the current position of the access identifier, reads the specified number of bytes, and moves the access identifier to the position immediately following the last accessed byte. Calling **Hread** with the `length=0` reads the entire data element. To reposition an access identifier before writing data, use **Hseek**.

If *length* is longer than the data element, the read operation is terminated at the end of the data element, and the number of read bytes is returned. Although only one access identifier is allowed per data element, it is possible to interlace reads from multiple data elements in the same file. It is assumed that *data* is large enough to hold the specified data length.

FORTRAN — None

Hseek

```
intn Hseek(int32 H_id, int32 offset, intn origin)
```

H_id	IN:	Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**
offset	IN:	Number of bytes to seek to from the origin
origin	IN:	Position to add the *offset* to; DF_START or DF_CURRENT

Purpose Sets the access pointer to an offset within a data element.

Return value Returns SUCCEED(0) if successful and FAIL(-1) otherwise.

Description Sets the seek position for the next **Hread** or **Hwrite** operation by moving an access identifier to the specified position in a data element. The *origin* argument, together with the *offset* argument, determine the byte location for the access identifier. If the *origin* is DF_START, the *offset* is added to the beginning of the data element. If *origin* is DF_CURRENT, the *offset* is added to the current position of the access identifier.

This routine fails if the access identifier if *H_id* is invalid or if the seek position is outside the range of the data element.

FORTRAN None

Hstartread

```
int32 Hstartread(int32 file_id, uint16 tag, uint16 ref)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag to search for
ref	IN:	Reference number to search for

Purpose Creates and positions a read access identifier in the data element identified by the specified tag and reference number.

Return value Returns the access identifier for a data element if successful and FAIL(-1) otherwise.

Description This function begins by searching the data descriptor list for the data element with the specified tag and reference number. All searches begin at the head of the data descriptor list.

If the search is successful, an access identifier is created and positioned at the beginning of the data element, otherwise an error is returned. Wildcards can be used for any *tag* or *ref* (DFTAG_WILDCARD, DFREF_WILDCARD). All access identifiers must be released through **Hendaccess** before closing the HDF file.

FORTRAN None

Hstartwrite

```
int32 Hstartwrite(int32 file_id, uint16 tag, uint16 ref, int32 length)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag to write to
ref	IN:	Reference number to write to
length	IN:	The length of the data element

Purpose Positions a write access identifier at the beginning of a new or existing data element.

Return value Returns the access identifier for a data object if successful and `FAIL(-1)` otherwise.

Description **Hstartwrite** begins by searching the data descriptor list for the specified *tag* and *ref*. If the tag and reference number exist, **Hstartwrite** positions an access identifier at the beginning of the existing data element. Data in the existing data element is not overwritten until a call is made to **Hwrite**. If the tag and reference number are not found, **Hstartwrite** allocates space for a new data object of length *length* and positions an access identifier at the head of the data element.

FORTRAN None

Hsync

```
intn Hsync(int32 file_id)
```

file_id	IN:	HDF file identifier returned by **Hopen**

Purpose Synchronizes on-disk HDF file with the memory representation.

Return value Returns `SUCCEED(0)` if successful and `FAIL(-1)` otherwise.

Description This routine is currently vacuous as the on-disk representation of an HDF file is always the same as its in-memory representation. However, future releases of the HDF library may employ buffering schemes, so this might not always be the case. **Hsync** will be provided to force the two representations to become consistent.

FORTRAN None

Htrunc

```
int32 Htrunc(int32 H_id, int32 trunc_len)
```

H_id	IN:	Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**
trunc_len	IN:	Length to truncate element

Purpose	Truncates a data element to a specified length
Return value	Returns the length of a data element if found and `FAIL(-1)` otherwise.
Description	Truncates the data object specified by the *H_id* to the length *trunc_len*. This routine is used primarily for resizing data elements containing compressed data.
FORTRAN	None

Hwrite

```
int32 Hwrite(int32 H_id, int32 length, uint8 *data)
```

H_id	IN:	Access identifier returned by **Hstartwrite**
len	IN:	Length of segment to write
data	IN:	Pointer to the data to write

Purpose — Writes the next data segment to a specified data element.

Return value — Returns the length of the segment actually written if successful and `FAIL(-1)` otherwise.

Description — **Hwrite** begins writing at the current position of the access identifier, writes the specified number of bytes, and moves the access identifier to the position immediately following the last accessed byte. Calling **Hwrite** with the `length=0` returns an error. To reposition an access identifier before writing data, use **Hseek**.

If the space allocated in the data element is smaller than the length of *data*, the data is truncated to the length of the data element. Although only one access identifier is allowed per data element, it is possible to interlace writes to multiple data elements in the same file.

FORTRAN — None

HXcreate

```
int32 HXcreate(int32 file_id, uint16 tag, uint16 ref, char *extern_name,
          int32 offset, int32 length)
```

file_id	IN:	HDF file identifier returned by **Hopen**
tag	IN:	Tag for the external element
ref	IN:	Reference number for the external element
extern_name	IN:	Name of the external file
offset	IN:	Byte offset for the external element in the external file
length	IN:	Length of the external element

Purpose Creates an external element.

Return value Returns the access identifier for the external element if successful and FAIL(-1) otherwise.

Description An external element is a data element stored in a different file than its data descriptor. **HXcreate** can create new external elements, move existing data elements from and HDF file to an external file, or incorporate data from an external file into an HDF file.

HXcreate creates a new external element if given an unused tag and reference number and the file name of a nonexistent file. Specify the *offset* as the number of bytes between the beginning of the external file and the beginning of the external element. After creating the new external element, use **Hwrite** to write data to it.

HXcreate moves a data element from an HDF file to an external file if given an existing tag and reference number. During this operation, the original data element is deleted from the HDF file and copied into the external file. Any data located at the specified offset in the external file is overwritten. Specify the *offset* as if creating a new external element. The *length* argument is ignored.

HXcreate incorporates data from an external file into an HDF file if given and new tag and reference number and the file name of an existing external file. This operation does not modify the external file, it simply creates a data descriptor in the specified HDF file that points to the external data. The *offset* and *length* should reflect the location and length of the data in the external file.

Moving existing data elements to external files will not automatically close open access identifiers. Open access identifiers prevent a file from closing, therefore it is important to use **Hendaccess** to remove any open identifiers before attempting to close a file.

Distributing a single data element over multiple external files is not supported.

FORTRAN None