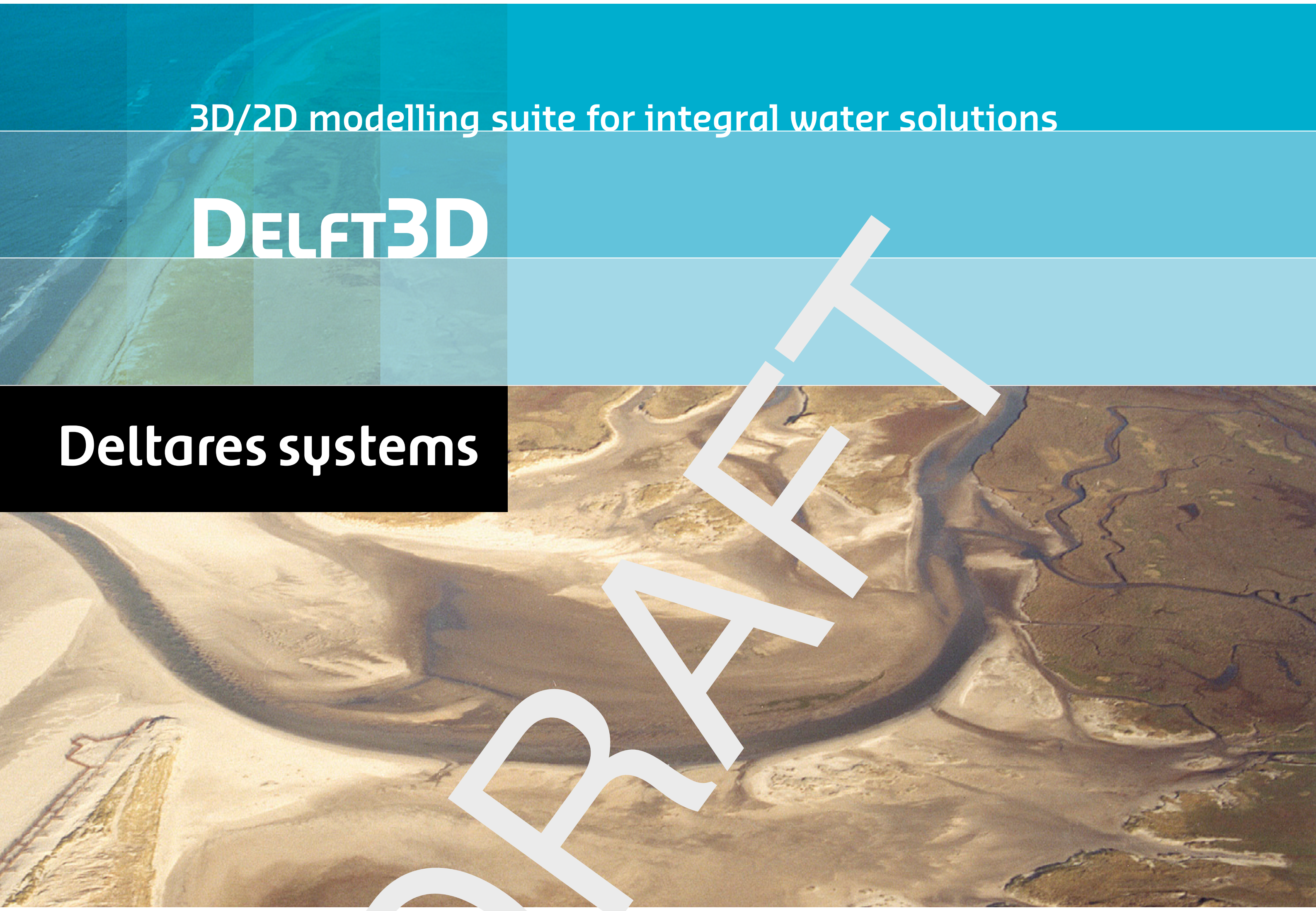


3D/2D modelling suite for integral water solutions

Delft3D

Deltares systems

NEFIS

Deltares

Enabling Delta Life

User Manual

NEFIS Library

Neutral File System for data storage and retrieval

User Manual

Version: 5.00
SVN Revision: 50387

April 18, 2018

NEFIS Library, User Manual

Published and printed by:

Deltares
Boussinesqweg 1
2629 HV Delft
P.O. 177
2600 MH Delft
The Netherlands

telephone:	+31 88 335 82 73
fax:	+31 88 335 85 82
e-mail:	info@deltares.nl
www:	https://www.deltares.nl

For sales contact:

telephone:	+31 88 335 81 88
fax:	+31 88 335 81 11
e-mail:	software@deltares.nl
www:	https://www.deltares.nl/software

For support contact:

telephone:	+31 88 335 81 00
fax:	+31 88 335 81 11
e-mail:	software.support@deltares.nl
www:	https://www.deltares.nl/software

Contents

List of Tables

1 Introduction

1.1 General

NEFIS is a library of functions designed for scientific programs. These programs are characterised by their large volume of input and output data. NEFIS is able to store and retrieve large volumes of data on file or in shared memory. To achieve a good performance when storing and retrieving data, the files are self-describing binary direct access files. Furthermore one of the array dimensions may be variable and the sequence on the file can be prescribed. NEFIS also allows users to store data in a machine-independent way on files, which means that the data files can be interchanged between computer systems without having to be converted. Data within NEFIS is divided into a hierarchical structure of groups, cells and elements. This hierarchical structure is used to find the location in the file where the data should be stored or retrieved. An element is the smallest unit which can be accessed at one time. One or more elements make up a cell; and a group is defined as one or more dimensional arrays of cells. This shows the logical cohesion of the data to be represented. Flags (in this context referred as attributes) can be attached to groups as desired. These attributes can, for example, define a match between groups. They may also contain superscripts and subscripts for graphic design. NEFIS can exist of one file for input and retrieval of data (i.e. a definition and a data part). The previous NEFIS version needed two files for input and retrieval of data (i.e. a data file and a definition file). A data file contains the data supplied by the user and the attributes that have been added. The definition file contains the description of the structure. The relationship between a data file and a definition file is determined by the application. This means that one definition file can be used by various data files. The opposite is also possible (i.e. a data file can be used from different definition files). More over, a well-defined definition file is able to scope all data files of a company.

1.2 Comments on the use of NEFIS

In almost all cases calling up a NEFIS function involves the use of a file descriptor. This file descriptor is a integer value which is used by the NEFIS function to determined which files need to be accessed. All NEFIS functions return an integer$*$4 value representing an error code. In the description of the functions, FORTRAN 77 notation is used to indicate the length of arrays. An array A with N elements is therefore given as follows: $A(1:N)$. A ’dummy’ must be entered for a parameter in several functions. The value represents the size of the memory that is available to store data. Not all data is written to the files after every operation; parts are buffered. The functions Flsdef and Flsdat are available to compel these buffers to be written to the file. It is then prudent to call these functions up after writing data. The function Clsnef writes these buffers to the files depending on the access type, the functions Clsdat and Clsdef also write these buffers to the files. The Cldtnf and Cldfnf functions do not write these buffers to the files and therefor the timestamp will not be changed if nothing is written to file. The functions Clsdat, Clsdef, Cldtnf and Cldfnf should be used if the functions Opndat and Opndef are used. To overcome the problem of two types of functions to close the NEFIS files the combination of Crenef and Clsnef function should be used, these functions take into account the accesstype of the files.

2 Definitions

2.1 Element definition

An element is the smallest unit, which can be accessed at one time. An element is a single variable, or one or more dimensional array of single variables with fixed limits. The type, dimensions, etc., of the element are fixed in a so-called element definition which is being stored in the definition file.

The definition of an element consists of:

- a unique name
- the type of the element
- the size of the single variable (in bytes) of which the element is made up
- the number of dimensions (0 for a single variable)
- the dimensions (for an array of single variables)
- meta data which state the quantity, unit and the description of the element

The following "basic" types are permitted in NEFIS (given for C and Fortran 77):

Table 2.1: *Supported basic types*

Size [bytes]	C	Fortran
4	Float	REAL∗4
8	Double	REAL∗8
2	Short	INTEGER∗2
4	Int	INTEGER∗4
8	Long	–
2	Short	LOGICAL∗2
4	Int	LOGICAL∗4
1	Char	CHARACTER∗1
2 ∗ 4	2 ∗ float	COMPLEX∗8
2 ∗ 8	2 ∗ double	COMPLEX∗16

2.2 Cell definition

A cell consists of combination of one or more elements whose type may differ. This combination is fixed in a so-called cell definition that is stored in the definition file/part. In the cell definition the elements which make up the cell are fixed as well as their storage sequence. Once a cell has been defined it has a fixed size. Each cell definition has a unique name.

2.3 Group definition

A group consists of one or more dimensional arrays of cells. One of the dimensions may be variable. The description of the group is fixed in a so-called group definition, which is stored in the definition file/part. The storage sequence needs to be defined in the case of multi-dimensional structures. This storage sequence defines which index must count first (most rapidly), which index must count second, etc.

3 Data handling

3.1 Accessing data

Data is written to and retrieved from file by means of a group name. Before data can be accessed, the group name must be defined on the data file. At that moment space is reserved for the data and for a number of attributes. NEFIS allows you to define several different group names with the same group definition. When the group has been defined, the data can be accessed using the group name and the element name as defined in the cell. The desired dimensions of a group can be specified. Remember that an element is the smallest unit that can be accessed. A buffer is used when data is stored or retrieved. The physical arrangement of data in this buffer must correspond with the way in which the cells are defined. When for example a cell is defined with REAL velocity components (u, v, w), then the values for the elements of this cell must also appear in the same order in the buffer. The type of buffer is of no importance for NEFIS functions and is determined by the user. For this reason, in the description of functions the type of buffer is given as "void". Data can be written and read. Data can also be altered (overwritten), but cannot be removed. Definitions can be written and read, but cannot be altered (overwritten) or removed.

3.2 Data conversion

Where files have to be used on several different computer systems, they may be stored in a standardised neutral (ANSI/IEEE 754) representation using xdr-routines if the variables occupy more than 4 bytes. Characters (CHARACTER∗1) are written directly to file. Integers and logicals (INTEGER∗2 and LOGICAL∗2) are handled by a NEFIS routine. This is done by giving the value "N" when the files are created with function "Crenef" via the CODING parameter. This enables the data to be correctly converted when written. If the data is subsequently entered into another system, it is converted back into the relevant machine representation. If the coding parameter is not a "N" while creating the NEFIS files the data will not be converted and is written directly to file. Proposed coding value is than "B" (binary). When reading a NEFIS the best choice is to assign a blank (" ") as coding value, this means that the coding type is read from file.

3.3 Error handling

All NEFIS functions are INTEGER∗4 functions. The functions give a return value equal to 0 when the function has been completed without an error. If the value is not equal to 0, the function has ended abnormally and the corresponding error message can be obtained with the function Neferr.

4 API to NEFIS functions

The functions currently available in NEFIS to manipulate descriptions and data are given below in alphabetical order:

Name	Definition
4.1	Close a NEFIS data and definition file
4.2	Declare a data group in the data file
4.3	Create a NEFIS data and definition file
4.4	Define a cell
4.5	Define an element
4.6	Define a group
4.7	Flush buffers to the data file
4.8	Flush buffers to the definition file
4.9	Read one or all string elements from the data file
4.9	Read one or all alpha numeric elements from the data file
4.10	Give information from the header of a NEFIS definition file
4.11	Give information from the header of a NEFIS data file
4.12	Read an integer attribute from the data group
4.14	Read a real attribute from the data group
4.15	Read a character attribute from the data group
4.16	Read a cell definition
4.17	Read a group definition name from a data group
4.18	Read an element definition
4.19	Read the first cell name on a definition file
4.20	Read the first element from the definition file
4.21	Read the first group from the definition file
4.22	Read the first name and corresponding value of the integer attributes of a given data group
4.23	Read the first name and corresponding value of the real attributes of a given data group
4.24	Read the first name and corresponding value of the character attributes of a given data group
4.25	Read the first data group name and definition name from the data file
4.26	Read the group definition
4.27	Determine the maximum used index of a data group with free dimension
4.19	Read the next cell name on the definition file
4.20	Read the first element from the definition file
4.21	Read the next group from the definition file
4.22	Read the next name and corresponding value of the integer attributes of a given data group
4.23	Read the next name and corresponding value of the real attributes of a given data group
4.24	Read the next name and corresponding value of the character attributes of a given data group
4.25	Read the next data group name
4.28	Retrieve an error message
4.29	Write one or more character elements to the data file
4.29	Write one or more elements to the data file
4.30	Write an integer attribute to the data group
4.31	Write a real attribute to the data file
4.32	Write a character attribute to the data file

The following sections contain a detailed description of NEFIS functions.

4.1 Clsnef

Description

Close the data and definition file. Depending on the access type the output buffers will be written to the files before closing the files.

Syntax

error = Clsnef (fd_nefis)

Parameters

fd_nefis		input/output	
	type	c	int $*$
		Fortran	integer$*$4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
	output	Set to -1, if function is successful	
error		return value	
	type	c	int
		Fortran	integer$*$4
	$= 0$	No error occurred	
	$\neq 0$	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.2 Credat

Description

Declare a data group in the data file.

Syntax

error = Credat (fd_nefis, grpnam, grpdef)

Parameters

fd_nefis		input	
	type	c	int
		Fortran	integer$*$4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char $*$
		Fortran	character$*$16
	input	Name of the data group to be created. This name must be unique to the data file.	
grpdef		input	
	type	c	char $*$
		Fortran	character$*$16

	input	Name of the group definition to be used for this data group. This group definition must already be defined on the definition file (see function Defgrp 4.6).	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	$\neq 0$	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remarks:

- ◇ Space is reserved on the data file, but this space is not initialised.
- ◇ For data groups with a variable dimension the space is only then reserved when the data is being written.

4.3 Crenef

Description
Create or open a NEFIS data and definition file

Syntax
error = Crenef (fd_nefis, dat_name, def_name, coding, access)

Parameters

fd_nefis		input/output	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor	
	output	Set to -1, if function is unsuccessful	
dat_name		input	
	type	c	char ∗
		Fortran	character(len=∗)
	input	Name of the data file, maximum file name length is 256 characters.	
def_name		input	
	type	c	char ∗
		Fortran	character(len=∗)
	input	Name of the definition file, maximum file name length is 256 characters.	
coding		input	
	type	c	char[1]
		Fortran	character(len=1)
	= 'B'	big-endian representation of the data, we called it neutral representation	
	= 'L'	little-endian representation of the data	
	≠ 'B' or 'L'	Endianess-representation is taken from the machine	

If the data file does not exist, this is an input parameter stating whether the data in the file have to be converted into the big-endian (IEEE-754) representation. Where the data file does already exist, this is an output parameter stating whether or not the data in the file are already in neutral representation.

access		input	
	type	c	char
		Fortran	character(len=1)
	$=$ 'c'	create a NEFIS file set, existing NEFIS file will be deleted	
	$=$ 'r'	read only access of the NEFIS file set	
	$=$ 'u'	update of the NEFIS file set, write and read access of the NEFIS file set.	

error		return value	
	type	c	int
		Fortran	integer$*$4
	$=0$	No error occurred	
	$\neq 0$	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.4 Defcel

Description

Define a cell

Syntax

error = Defcel (fd_nefis, celnam, nelems, elmnms)

Parameters

fd_nefis		input	
	type	c	int
		Fortran	integer$*$4
	input	NEFIS file descriptor (see function Crenef, 4.3)	

celnam		input	
	type	c	char $*$
		Fortran	character(len=16)
	input	Name of the cell to be defined.	

nelems		input	
	type	c	int $*$
		Fortran	integer$*$4
	input	Number of elements in this cell.	

elmnms		input	
	type	c	int $*$
		Fortran	integer$*$4

	input	Array(1:NELEMS) with the names of the elements which make up this cell. The element names must already be defined. (see function Defelm 4.5). The sequence of the names in this array is also the physical order in which the data in the buffer are expected when reading and writing entire cells.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.5 Defelm

Description

Define an element.

Syntax

error = Defelm (fd_nefis, elmnam, elmtyp, nbytsg, elmqty, elmunt, elmdes, elmndm, elmdms)

Parameters

fd_nefis		input	
	type	c	int
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
elmnam		input	
	type	c	char ∗
		Fortran	character(len=16)
	input	Name for the element to be defined.	
elmtyp		input	
	type	c	char ∗
		Fortran	character(len=8)
	input	Type of element. The following types can be used: REAL, INTEGER, LOGICAL, CHARACTE(R) and COMPLEX.	
nbytsg		input	
	type	c	int
		Fortran	integer∗4
	input	The size in bytes of a single variable of this type. For example, this is usually 4 for an element that will contain real values.	
elmqty		input	
	type	c	char ∗
		Fortran	character(len=16)
	input	The quantity of this element.	
elmunt		input	

	type	c	char ∗
		Fortran	character(len=16)
	input	The unit of this element.	
elmdes		input	
	type	c	char ∗
		Fortran	character(len=64)
	input	The description of this element.	
elmndm		input	
	type	c	int
		Fortran	integer∗4
	input	The number of dimensions of this element. This is 0 for a single element. For elements other than single elements, the number of dimensions may be from 1 to 5.	
elmdms		input	
	type	c	int ∗
		Fortran	integer∗4
	input	This is an array(1:ELMNDM) with the dimension for an element other than a single element. The sequence and the size of the values in this array determine the structure of the element. For a single element, this value may be 0. The dimensions must be larger than 0 for an array. The correctness of the dimensions is not being checked.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.6 Defgrp

Description

Define a group on the definition file.

Syntax

error = Defgrp (fd_nefis, grpnam, celnam, grpndm, grpdms, grpord)

Parameters

fd_nefis		input	
	type	c	int
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char ∗
		Fortran	character(len=16)

	input	Name for the group to be defined.	
celnam		input	
	type	c	char ∗
		Fortran	character(len=16)
	input	Name of the cell with which this group is made up. This cell must already be defined (see function Defcel 4.4).	
grpndm		input	
	type	c	int ∗
		Fortran	integer∗4
	input	The number of dimensions of this group. The number of dimensions may be 0 for a group with only 1 cell. The maximum number of dimensions is 5.	
grpdms		input	
	type	c	int ∗
		Fortran	integer∗4
	input	This is an array(1:GRPNDM) with the dimensions. One of the dimensions may be 0 which number indicates the part that is variable. The correctness of the dimensions is not being checked.	
grpord		input	
	type	c	int ∗
		Fortran	integer∗4
	input	This is an array(1:GRPNDM) which gives the order in which the cells must be written to the data file (which index runs most rapidly, which runs next most rapidly etc.). The sequence which operates in FORTRAN for a multi-dimensional array is 1, 2, 3, etc.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ The group order array (grpord) allows you to write data to a file in a several different order. The best possible performance is achieved by selecting a sequence which is the same as that in which the data will usually be retrieved later. For example, where it is necessary to store precipitation observations which are made on a daily basis in various locations and where the pattern of access to these observations is normally that all locations call them up at a specific time, it is advisable to arrange for the counter giving the locations to open more rapidly than the others. This parameter is dummy where the number of dimensions is 0 or 1.

4.7 Flsdat

Description

Flush the buffers to the data file.

Syntax
error = Flsdat (fd_nefis)

Parameters

fd_nefis		input	
	type	c	int $*$
		Fortran	integer$*$4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
error		return value	
	type	c	int
		Fortran	integer$*$4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ It is prudent to use this function after writing data to the file, because otherwise if the application ends abnormally the data is present on the file but can no longer be accessed. The Clsnef function also writes the buffers to the file.

4.8 Flsdef

Description
Flush the buffers to the definition file.

Syntax
error = Flsdef (fd_nefis)

Parameters

fd_nefis		input	
	type	c	int $*$
		Fortran	integer$*$4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
error		return value	
	type	c	int
		Fortran	integer$*$4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ It is prudent to use this function after writing the definition file, because otherwise if the application ends abnormally the definitions is present on the file but can no longer be accessed. The Clsnef function also writes the buffers to the file.

4.9 Getels/Getelt

Description

Read one or all (same kind) elements from a group on a data file. Getels is used for string values and Getelt is used for alpha-numeric values.

Syntax

error = Getels (fd_nefis, grpnam, elmnam, uindex, usrord, buflen, buffer)
error = Getelt (fd_nefis, grpnam, elmnam, uindex, usrord, buflen, buffer)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the data group of which one or more elements must e read.	
elmnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the element which is to be read. When Elmnam = '∗', this means that all elements must be read.	
uindex		input	
	type	c	int ∗ Uindex[<grpndm>][3]
		fortran	integer Uindex(3,<GRPNDM>)
	input	This array contains information on the indices of the cells that must be run. For each dimension of the data group three numbers must be given, namely the start index, the end index and the step size.	
usrord		input	
	type	c	int ∗ Usrord[<grpndm>]
		fortran	integer Usrord(<GRPNDM>)
	input	This is an array (1:GRPDNM) which determines the sequence in which the cells must be run. This is achieved by stating which index runs most rapidly, which runs next most rapidly etc.) With this the "view" of a group can be changed provided that also the buffer is filled in this sequence.	
buflen		input	
	type	c	int ∗
		fortran	integer
	input	Size in bytes of the available buffer.	
buffer		output	
	type	c	void
		fortran	void

	output	Buffer in which the values of the cells concerned are written (strings and numbers). When the buffer is used to receive strings, it also possible to use the function Getels.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remarks:

- If a character string/array is retrieved by a c/c++ program no '$\backslash 0$' is added to the string or array item. This is because the NEFIS library does not recognize the single item, it is just reading the bites.

- How parameter UINDEX works can best be presented by an example: Suppose you have defined with Defgrp a group with dimensions (20,13) and reversed the array with parameter USRORD towards (13, 20). With the following values in UINDEX:

```
UINDEX(1-3,1) = 1, 13, 3
UINDEX(1-3,2) = 5, 20, 5
```

  then the cells will be run in the following sequence:

```
[ 5,1] [ 5,4] [ 5,7] [ 5,10] [ 5,13]
[10,1] [10,4] [10,7] [10,10] [10,13]
[15,1] [15,4] [15,7] [15,10] [15,13]
[20,1] [20,4] [20,7] [20,10] [20,13].
```

  The given indices must be within the defined limits. Obviously, for indices of the variable dimension (see Defgrp 4.6) this restriction does not hold. The step size must be positive.

4.10 Gethdf

Description

Give information from the header of a definition file.

Syntax

c: error = Gethdf (fd_nefis, header)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
header		input	
	type	c	char ∗ [128+1]
		fortran	character∗128

	input	In this string the information from the file header is being put. Among other things this information contains the name of the system upon which the file is being created and whether or not the data representation is neutral (ANSI/IEEE 754). When using two separated NEFIS files for data and definition the size of the header can be 60 instead of 128.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ If the header is retrieved by a c/c++ program no '\0' is added to the string

4.11 Gethdt

Description
Give information from the header of a data file.

Syntax
error = Gethdt (fd_nefis, header)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
header		input	
	type	c	char ∗
		fortran	character∗128
	input	In this string information from the file header is being put. Among other things this information contains the name of the system upon which the file is being created and whether or not the data representation is neutral (ANSI/IEEE 754). When using two separated NEFIS files for data and definition the size of the header can be 60 instead of 128.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

- ◇ If the header is retrieved by a c/c++ program no ’\0’ is added to the string

4.12 Getiat

Description

Read an integer attribute from a data group.

Syntax

error = Getiat (fd_nefis, grpnam, attnam, attval)

Parameters

fd_nefis		input	
	type	c	int *
		Fortran	integer*4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char *
		fortran	character*16
	input	Name of the data group from which an integer attribute is to be read.	
attnam		input	
	type	c	char *
		fortran	character*16
	input	Name of the attribute (see function Putiat 4.30).	
attval		output	
	type	c	int *
		Fortran	integer*4
	input	Attribute value.	
error		return value	
	type	c	int
		Fortran	integer*4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

- ◇ See also the functions Inqfia and Inqnia 4.22.

4.13 Getnfv

Description
Returns the full version number of the NEFIS library.

Syntax
error = Getnfv (nef_version)

Parameters

nef_version		output	
	type	c	char $\ast\ast$
		fortran	character$\ast$
	input	In this string the full version number of the NEFIS library is returned.	
error		return value	
	type	c	int
		Fortran	integer$\ast$4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

The corresponding error message can not be retrieved with the function Neferr.

Example:

```
@(#)Deltares, NEFIS Version 5.07.02.3965M, Aug 4 2014, 20:09:15
```

4.14 Getrat

Description
Read a real attribute from a data group.

Syntax
error = Getrat (fd_nefis, grpnam, attnam, attval)

Parameters

fd_nefis		input	
	type	c	int $\ast$
		Fortran	integer$\ast$4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char $\ast$
		fortran	character$\ast$16
	input	Name of the data group from which an integer attribute is to be read.	
attnam		input	
	type	c	char $\ast$
		fortran	character$\ast$16
	input	Name of the attribute (see function Putrat 4.31).	

attval		output	
	type	c	float ∗
		fortran	real∗4
	output	Attribute value.	

error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ See also the functions INQFRA and INQNRA.

4.15 Getsat

Description

Read a string attribute from a data group.

Syntax

error = Getsat (fd_nefis, grpnam, attnam, attval)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	

grpnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the data group from which an integer attribute is to be read.	

attnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the attribute (see function Putrat 4.31).	

attval		output	
	type	c	char ∗
		fortran	character∗16
	output	Attribute value.	

error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remarks:

- If string attribute is retrieved by a c/c++ program no ’$\backslash 0$’ is added to the string
- See also the functions Inqfsa and Inqnsa 4.24.

4.16 Inqcel

Description

Read a cell definition from the definition file

Syntax

error = Inqcel (fd_nefis, celnam, nelems, elmnms)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
celnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the cell to be defined.	
nelems		input/output	
	type	c	int ∗
		Fortran	integer∗4
	input	current length of array ELMNMS, value will be checked against number of elements in this cell definition	
	output	the number of elements used in this cell definition	
elmnms		output	
	type	c	char ∗ elmnms[nelems][16+1]
		fortran	character∗16 elmnms(nelems)
		An array (1:NELEMS) with the names of the elements used in this cell definition.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.17 Inqdat

Description

Read corresponding group definition from the data group.

Syntax

error = Inqdat (fd_nefis, grpnam, grpdef)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the data group of which the definition name is to be read.	
grpdef		output	
	type	c	char ∗
		fortran	character∗16
	output	Name of the definition used for this data group.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.18 Inqelm

Description

Read an element definition from the definition file.

Syntax

error= Inqelm (fd_nefis, elmnam, elmtyp, nbytsg, elmqty, elmunt, elmdes, elmndm, elmdms)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
elmnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the element whose definition is to be read.	

elmtyp		output	
	type	c	char $*$
		fortran	character$*$8
	output	Type of element.	
nbytsg		output	
	type	c	int
		Fortran	integer$*$4
	output	The size in bytes of a single variable of the element.	
elmqty		output	
	type	c	char $*$
		fortran	character$*$16
	output	The quantity of this element.	
elmunt		output	
	type	c	char $*$
		fortran	character$*$16
	output	The unit of this element.	
elmdes		output	
	type	c	char $*$
		fortran	character$*$64
	output	The description of this element.	
elmndm		input/output	
	type	c	int $*$
		Fortran	integer$*$4
	input	Current length of array ELMDMS, value will be checked against number of dimensions of this element. Because ELMNDM is not known in advance, it is recommended that the current length of this array be set at the maximum value (=5).	
	output	The number of dimensions of this element (maximum 5).	
elmdms		output	
	type	c	int
		Fortran	integer$*$4
	output	This is an array(1:ELMNDM) with the dimension for an element other than a single element.	
error		return value	
	type	c	int
		Fortran	integer$*$4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.19 Inqfcl/Inqncl

Description

Read the first and next cell name, dimension, size in bytes and containing element names from the definition file.

Syntax

error = Inqfcl (fd_nefis, celnam, nelems, bytes, elmnms)
error = Inqncl (fd_nefis, celnam, nelems, bytes, elmnms)

Parameters

fd_nefis		input	
	type	c	int *
		Fortran	integer*4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
celnam		output	
	type	c	char *
		fortran	character*16
	output	Name of the cell for which data is retrieved.	
nelems		output	
	type	c	int *
		Fortran	integer*4
	input	Current length of array ELMNMS, value will be checked against number of elements in this cell definition.	
	output	The number of elements used in this cell definition.	
bytes		output	
	type	c	int *
		Fortran	integer*4
	output	Length of the cell in bytes.	
elmnms		output	
	type	c	char * elmnms[nelems][16+1]
		fortran	character*16 elmnms(nelems)
	output	Array(1:NELEMS) with the names of the elements which make up this cell. The element names must already be defined. (see function Defelm 4.5) The sequence of the names in this array is also the physical order in which the data in the buffer are expected when reading and writing entire cells.	
error		return value	
	type	c	int
		Fortran	integer*4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ The elmnms array is a character array, the ’\0’ is not added to the array items when retrieving the array.

4.20 Inqfel/Inqnel

Description

Read the first and next element from the definition file.

Syntax

error = Inqfel (fd_nefis, elmnam, elmtyp, elmqty, elmunt, elmdes, bytes, numbyt, elmndm, elmdms)
error = Inqnel (fd_nefis, elmnam, elmtyp, elmqty, elmunt, elmdes, bytes, numbyt, elmndm, elmdms)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
elmnam		output	
	type	c	char ∗
		fortran	character∗16
	output	Name of the element.	
elmtyp		output	
	type	c	char ∗
		fortran	character∗8
	output	Type of element. An element can be one of the following types: REAL, INTEGER, LOGICAL, CHARACTE(R) or COMPLEX.	
elmqty		output	
	type	c	char ∗
		fortran	character∗16
	output	The quantity of the element.	
elmunt		output	
	type	c	char ∗
		fortran	character∗16
	output	The unit of the element.	
elmdes		output	
	type	c	char ∗
		fortran	character∗64
	output	The description of the element.	
bytes		output	
	type	c	int ∗
		Fortran	integer∗4
	output	The size in bytes of a single variable of this type. For example, this is usually 4 for an element that will contain real values.	
numbyt		output	
	type	c	int ∗
		Fortran	integer∗4

	output	The size in bytes of the complete element. That is usually bytes∗elmdms(1)∗...∗elmdms(elmndm) for an element which will contain real values.	
elmndm		output	
	type	c	int ∗
		Fortran	integer∗4
	input	current length of array ELMDMS, value will be checked against number of dimensions of this element. Because ELMNDM is not known in advance, it is recommended that the current length of this array be set at the maximum value (=5).	
	output	the number of dimensions of the element. The number of dimensions may be from 1 to 5.	
elmdms		output	
	type	c	int ∗
		Fortran	integer∗4
	output	This is an array(1:ELMNDM) with the dimension for an element. The sequence and the size of the values in this array determine the structure of the element.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.21 Inqfgr/Inqngr

Description

Read the first or next group from definition file.

Syntax

error = Inqfgr (fd_nefis, grpnam, celnam, grpndm, grpdms, grpord)
error = Inqngr (fd_nefis, grpnam, celnam, grpndm, grpdms, grpord)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		output	
	type	c	char ∗
		fortran	character∗16
	output	Name of the group.	
celnam		output	
	type	c	char ∗

		fortran	character∗16
	output	Name of the cell with which this group is made up. (see function Defcel, 4.4).	
grpndm		input/output	
	type	c	int ∗
		Fortran	integer∗4
	input	current length of array GRPDMS and GRPORD, this value will be checked against number of dimensions of this group. Because GRPNDM is not known in advance, it is recommended that the current length of this array be set at the maximum value (=5).	
	output	The number of dimensions of this group. The maximum number of dimensions is 5.	
grpdms		output	
	type	c	int ∗
		Fortran	integer∗4
	output	This is an array(1:GRPNDM) with the dimensions. One of the dimensions may be 0 which number indicates the part that is variable. The correctness of the dimensions is not being checked.	
grpord		output	
	type	c	int ∗
		Fortran	integer∗4
	output	This is an array(1:GRPNDM) which gives the order in which the cells must be written to the data file (which index runs most rapidly, which runs next most rapidly etc.). The sequence which operates in FORTRAN for a multi-dimensional array is 1, 2, 3, etc..	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.22 Inqfia/Inqnia

Description

Read the names and values of the first and next integer attributes, INQFIA and INQNIA respectively, of a data group.

Syntax

error = Inqfia (fd_nefis, grpnam, attnam, attval)
error = Inqnia (fd_nefis, grpnam, attnam, attval)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4

	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the data group of which the names and values of the attributes must be read.	
attnam		output	
	type	c	char ∗
		fortran	character∗16
	output	Name of the attribute (see function Putiat 4.30).	
attval		output	
	type	c	int ∗
		Fortran	integer∗4
	output	Attribute value.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

- ◇ The function INQFIA gives the first name and the corresponding value. With function INQNIA possibly following names and values may be retrieved.

4.23 Inqfra/Inqnra

Description

Read the names and values of the first and next real attributes, INQFRA and INQNRA respectively, of a data group.

Syntax

error = Inqfra (fd_nefis, grpnam, attnam, attval)
error = Inqnra (fd_nefis, grpnam, attnam, attval)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the data group of which the names and values of the attributes must be read.	

attnam		output	
	type	c	char ∗
		fortran	character∗16
	output	Name of the attribute (see function Putrat 4.31).	

attval		output	
	type	c	int ∗
		Fortran	integer∗4
	output	Attribute value.	

error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ The INQFRA function gives the first name and corresponding value. With the INQNRA function the possibly following names and values may be retrieved.

4.24 Inqfsa/Inqnsa

Description

Read the names and values of the first and next character attributes, INQFSA and INQNSA respectively, of a data group.

Syntax

error = Inqfsa (fd_nefis, grpnam, attnam, attval)
error = Inqnsa (fd_nefis, grpnam, attnam, attval)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	

grpnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the data group of which the names and values of the attributes must be read.	

attnam		output	
	type	c	char ∗
		fortran	character∗16
	output	Name of the attribute (see function Putsat, 4.32).	

attval		output	

	type	c	char ∗
		fortran	character∗16
	output	Attribute value.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$=0$	No error occurred	
	>0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ The INQFSA function gives the first name and corresponding value. With the INQNSA function the possibly following names and values may be retrieved.

4.25 Inqfst/Inqnxt

Description

Read the name of the first (INQFST) and the next (INQNXT) data group from a data file and the corresponding group definition name.

Syntax

error = Inqfst (fd_nefis, grpnam, grpdef)
error = Inqnxt (fd_nefis, grpnam, grpdef)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		output	
	type	c	char ∗
		fortran	character∗16
	output	Name of the first and the next data group, INQFST and INQNXT respectively, on the data file (see function Credat, 4.2).	
grpdef		output	
	type	c	char ∗
		fortran	character∗16
	output	Name of the definition used for this data group (see function Credat, 4.2).	
error		return value	
	type	c	int
		Fortran	integer∗4
	$=0$	No error occurred	
	>0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ INQFST gives the first name, INQNXT the next one until there are no names left.

4.26 Inqgrp

Description

Read a group definition from the definition file.

Syntax

error = Inqgrp (fd_nefis, grpdef, celnam, grpndm, grpdms, grpord)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpdef		input	
	type	c	char ∗
		fortran	character∗16
	input	Name for the group whose definition is to be retrieved.	
celnam		output	
	type	c	char ∗
		fortran	character∗16
	output	Name of the cell with which this group is made up.	
grpndm		input/output	
	type	c	int ∗
		Fortran	integer∗4
	input	The number of dimensions which can be stored in the arrays GRPDMS and GRPORD. Because GRPNDM is not known in advance, it is recommended to fix the current length of this array at the maximum value (=5).	
	output	The number of dimensions of this group.	
grpdms		output	
	type	c	int
		Fortran	integer∗4
	output	This is an array(1:GRPNDM) with the dimensions of the group.	
grpord		output	
	type	c	int
		Fortran	integer∗4
	output	This is an array (1:GRPNDM) which gives the order in which the data was written to the data file (which index runs most rapidly, which runs next most rapidly, etc.).	
error		return value	
	type	c	int

		Fortran	integer∗4
	$=0$	No error occurred	
	>0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.27 Inqmxi

Description

Give the maximum used index of the free dimension of a data groupp.

Syntax

error = Inqmxi (fd_nefis, grpnam, max_index)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	

grpnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name for the group whose maximum variable index is to be retrieved.	

max_index		output	
	type	c	int ∗
		Fortran	integer∗4
	output	Maximum index of group ’grpnam’. If the maximum index is zero then there is no data written to that group.	

error		return value	
	type	c	int
		Fortran	integer∗4
	$=0$	No error occurred	
	>0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

4.28 Neferr

Description

Error message can be retrieved from the NEFIS library and/or printed to standard output.

Syntax

error = Neferr (print_flag, error_string)

Parameters

print_flag		input	
	type	c	int
		Fortran	integer∗4
	input	Print flag to indicate whether the error message is send to standard output.	
		== 0	Do not print error string
		== 1	Do print the error string on standard output
		== 2	Do print the error string on standard error
error_string		output	
	type	c	char ∗
		fortran	character∗1024
	output	The string error_string contains the error message generated by the last NEFIS function which detected the error. Or if the last function did not detected an error the number of errors occurred when using the NEFIS library.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	

4.29 Putelt/Putels

Description

Write one or all (same kind) elements to a group on the data file. Putelt is used for alphanumeric values and Putels is used for string values.

Syntax

error = Putels (fd_nefis, grpnam, elmnam, uindex, usrord, buffer)
error = Putelt (fd_nefis, grpnam, elmnam, uindex, usrord, buffer)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char ∗ grpname[16+1]
		fortran	character∗16 grpnam
	input	Name of the data group of which one or all elements must be written.	
elmnam		input	
	type	c	char ∗ elmnam[16+1]
		fortran	character∗16 elmnam
	input	Name of the element which must be written. When Elmnam = '∗', this means that all (same kind) elements must be written.	
uindex		input	
	type	c	int ∗ Uindex[<grpndm>][3]
		fortran	integer Uindex(3,<GRPNDM>)

	input	This array contains information on the indices of the cells that must be walked through. For each dimension of the data group three numbers must be given viz. the start index, the end index and the step size.	
usrord		input	
	type	c	int ∗ Usrord[<grpndm>]
		fortran	integer Usrord (<GRPNDM>)
	input	This is an array (1:GRPNDM) which determines the sequence in which the cells must be stepped through. This is achieved by stating which index runs most rapidly, which runs next most rapidly etc.. This allows you to make up a desired group representation, provided that the buffer has the same sequence.	
buffer		output	
	type	c	void ∗
		fortran	void
	input	Buffer with values which must be written in the given element of the cells indicated by UINDEX. Obviously, the data in the buffer must have the same sequence.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ How parameter UINDEX works can best be presented by an example: Suppose you have defined with DEFGRP a group with dimensions (20,13) and reversed the array with parameter USRORD towards (13,20). With the following values in UINDEX:

```
UINDEX(1-3,1) = 1, 13, 3
UINDEX(1-3,2) = 5, 20, 5
```

then the cells will be run in the following sequence:

```
[ 5,1] [ 5,4] [ 5,7] [ 5,10] [ 5,13]
[10,1] [10,4] [10,7] [10,10] [10,13]
[15,1] [15,4] [15,7] [15,10] [15,13]
[20,1] [20,4] [20,7] [20,10] [20,13].
```

The given indices must be within the defined limits. Obviously, for indices of the variable dimension (see Defgrp 4.6) this restriction does not hold. The step size must be positive.

4.30 Putiat

Description

Write an integer attribute to a data group.

Syntax

error = Putiat (fd_nefis, grpnam, attnam, attval)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the data group at which an integer attribute is to be written.	
attnam		input	
	type	c	char ∗
		fortran	character∗16
	input	Name of the attribute.	
attval		input	
	type	c	int ∗
		Fortran	integer∗4
	input	Attribute value.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

- ◇ If this attribute name already exists then the previous value will be replaced by a new one.

4.31 Putrat

Description

Write a real attribute to a data group.

Syntax

error = Putrat (fd_nefis, grpnam, attnam, attval)

Parameters

fd_nefis		input	
	type	c	int ∗
		Fortran	integer∗4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char ∗
		fortran	character∗16

	input	Name of the data group at which an integer attribute is to be written.	
attnam		input	
	type	c	char *
		fortran	character*16
	input	Name of the attribute.	
attval		input	
	type	c	int *
		Fortran	integer*4
	input	Attribute value.	
error		return value	
	type	c	int
		Fortran	integer*4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

◇ If this attribute name already exists then the previous value will be replaced by a new one.

4.32 Putsat

Description

Write a string attribute to a data group.

Syntax

error = Putsat (fd_nefis, grpnam, attnam, attval)

Parameters

fd_nefis		input	
	type	c	int *
		Fortran	integer*4
	input	NEFIS file descriptor (see function Crenef, 4.3)	
grpnam		input	
	type	c	char *
		fortran	character*16
	input	Name of the data group at which an string attribute is to be written.	
attnam		input	
	type	c	char *
		fortran	character*16
	input	Name of the attribute.	
attval		input	
	type	c	char *

		fortran	character∗16
	input	Attribute value.	
error		return value	
	type	c	int
		Fortran	integer∗4
	$= 0$	No error occurred	
	> 0	Fatal error occurred	

Corresponding error message can be printed to standard error or retrieved with the function Neferr, 4.28.

Remark:

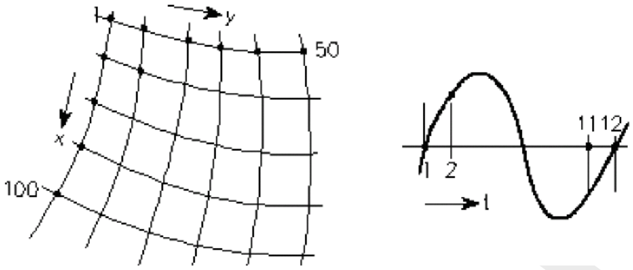
◇ If this attribute name already exists then the previous value will be replaced by a new one.

Deltares, 2016. “BIBTEX key with no entry, needed if no citations are made in the document.”

A Example of a data structure

Before data from an application can be written to NEFIS files, the structure in terms of elements, cells and groups must be determined. This can be done in various ways. For instance a field of 5 000 points (a 2-dimensional raster of 100×50) with at each point:

- ◇ a discharge Q
- ◇ a velocity V(2), consisting of two components
- ◇ waveheight spectrum Hs(10,4), 10 angles, 4 periods
- ◇ output data at 12 points in time



This data can be portrayed in various ways. Some options are described below.

Option A

Elements:	"DISCH"	Q
	"VELO"	V(2)
	"SPEC"	Hs(10,4)
Cell:	"POINT"	[DISCH, VELO, SPEC]
Group:	"MODEL"	(100, 50, 12) of type POINT

Option B

Elements:	"DISCH"	Q(100)
	"VELO"	V(100,2)
	"SPEC"	Hs(100,10,4)
Cell:	"CROSS-SECTION"	[DISCH, VELO, SPEC]
Group:	"MODEL"	(50, 12) of type CROSS-SECTION

Option C

Elements:	"DISCH"	Q(100,50)
	"VELO"	V(100,50,2)
	"SPEC"	Hs(100,50,10,4)
Cell:	"AREA"	[DISCH, VELO, SPEC]
Group:	"MODEL"	(12) of type AREA

The most important criterion when selecting one of the three possibilities is:
in which units data will need to be accessed subsequently (the larger the units, the better the performance) and in which units the data is available. The smallest unit which can be accessed is an element.

B Error/Warning numbers

Nr	Error/Warning description
1001	—
1002	Crenef: Data filename too long (length < %d)
1003	Crenef: Definition filename too long (length < %d)
1004	Gethdt: Supplied character string too small for header
1005	Gethdt: Unable to read data file header (file write only?)
1006	Gethdt: During reading of data file header
1007	Gethdf: Supplied character string too small for header
1008	Gethdf: Unable to read definition file header (file write only?)
1009	Gethdf: During reading of definition file header
1010	—
1011	Inqcel: User supplied array too small to contain Cell properties: '%s' %ld>%ld
1012	Inqelm: User supplied array too small to contain Element properties: '%s' %ld>%ld
1013	Inqfcl: User supplied array too small to contain Cell properties: '%s' %ld>%ld
1014	Inqncl: User supplied array too small to contain Cell properties: '%s' %ld>%ld
1015	Inqfgr: User supplied array too small to contain group properties: '%s' %ld>%ld
1016	Inqngr: User supplied array too small to contain group properties: '%s' %ld>%ld
1019	Putiat: Groupname '%s' or integer attribute name '%s' too long
1020	Putrat: Groupname '%s' or real attribute name '%s' too long
1021	Putsat: Groupname '%s' or real attribute name '%s' or attribute value '%s' too long
2001	—
2002	Crenef: Data filename too long (length < %d)
2003	Crenef: Definition filename too long (length < %d)
2004	Gethdt: Supplied character string too small for header
2005	Gethdt: Unable to read data file header (file write only?).
2006	Gethdt: During reading of data file header.
2007	Gethdf: Supplied character string too small for header
2008	Gethdf: Unable to read definition file header (file write only?).
2009	Gethdf: During reading of definition file header.
2010	—
2011	Getsat: User supplied attribute string too small
2012	Inqcel: Supplied array too small to contain all element names: '%s' %ld>%d
2013	Inqcel: User supplied array too small to contain Cell properties: '%s' %ld>%ld
2014	Inqdat: User supplied array to store group definition too small
2015	Inqelm: User supplied array's to store element definition too small: %s, %ld, %ld, %ld, %ld,%ld
2016	Inqelm: Element name '%s' too long
2017	Inqelm: User supplied array to contain element names too small
2018	Inqfst: User supplied array to contain names too small
2019	Inqelm: User supplied array's to store element definition too small
2020	Inqfel: User supplied array to contain element names too small
2021	Inqelm: User supplied array's to store element definition too small
2022	Inqfel: User supplied array to contain element names too small
2023	Inqfcl: Supplied array too small to contain all element names: '%s' %ld>%d
2024	Inqfcl: User supplied array too small to contain Cell properties: '%s' %ld>%ld
2025	Inqfcl: Supplied array too small to contain all element names: '%s' %ld>%d
2026	Inqncl: User supplied array too small to contain Cell properties: '%s' %ld>%ld
2027	Inqfgr: User supplied array to contain names too small
2028	Inqfgr: User supplied array too small to contain group properties: '%s' %ld>%ld

Nr	Error/Warning description
2029	Inqfgr: User supplied array to contain names too small
2030	Inqngr: User supplied array too small to contain group properties: ’%s’ %ld>%ld
2031	Inqfia: User supplied array to contain integer attribute names too small
2032	Inqfra: User supplied array to contain real attribute names too small
2033	Inqfsa: User supplied array to contain string attribute names too small
2034	Inqgrp: Group name too long ’%s’
2035	Inqgrp: User supplied array to contain group dimensions too small
2037	Inqnxt: User supplied array to contain names too small
2038	Inqnia: User supplied array to contain integer attribute names too small
2039	Inqnra: User supplied array to contain real attribute names too small
2040	Inqnra: User supplied array to contain string attributes (name/value) too small
2042	Putiat: Groupname ’%s’ or integer attribute name ’%s’ too long
2043	Putrat: Groupname ’%s’ or real attribute name ’%s’ too long
2044	Putsat: Groupname ’%s’ or string attribute name ’%s’ or attribute value ’%s’ too long
3001	Start value user index [%ld,2]=%ld should be smaller than [%ld,2]=%ld
3002	Increment value user index [%ld,2]=%ld should be greater than 0
3003	Start value user index [%ld,0]=%ld should be greater than zero
3004	Stop value %ld should be smaller than %ld
3005	Buffer length too small, should be %ld instead of %ld Group "%s", element "%s"
3006	Variable dimension %ld not found for: group "%s", element "%s"
4001	Start value user index [%ld,2]=%ld should be smaller than [%ld,2]=%ld
4002	Increment value user index [%ld,2]=%ld should be greater than 0
4003	Start value user index [%ld,0]=%ld should be greater than zero
4004	Stop value %ld should be smaller than %ld
5001	Number of dimensions not within the range [1,%d] Element ’%s’ has dimension %ld
5002	This size of real (%ld) is not supported
5003	This size of integer (%ld) is not supported
5004	This size of complex (%ld) is not supported
5005	This size of logical (%ld) is not supported
5006	This element type is not supported ’%s’
5007	Element ’%s’ already exists
5008	Error writing element ’%s’ to definition file
5009	Element ’%s’ does not exist
5010	Cel ’%s’ already exists
5011	Error writing cel ’%s’ to definition file
5012	—
5013	Cel ’%s’ does not exist
5014	Group ’%s’ already exists.
5015	Error on writing group ’%s’ to definition file.
5016	Number of dimensions not within the range [1,MAX_DIM] Group ’%s’
5017	Group definition ’%s’ does not exist.
5018	Cel ’%s’ does not exist.
5019	Data group ’%s’ already exists in data file
5020	Error on writing group ’%s’ with variable dimension.
5021	Error on writing group ’%s’ with fixed dimension
5022	Error on writing group ’%s’ with variable dimension.
5023	Error on writing group ’%s’ with fixed dimension
5024	Maximum size reached in DataDefinition file "%s"

Nr	Error/Warning description
5025	Maximum size reached in definition file "%s"
5026	Maximum size reached in definition file "%s"
5027	Maximum size reached in definition file "%s"
5028	Maximum size reached in definition file "%s"
5029	Maximum size reached in definition file "%s"
5030	Maximum size reached in DataDefinition file "%s"
5031	Maximum size reached in definition file "%s"
6001	Hashtable not written to data file ’%s’
6002	Hashtable not written to definition file ’%s’
6002	Hashtable not written to DefinitionData file ’%s’
6003	Cell ’%s’ does not exist in definition file
6004	Group ’%s’ does not exist in data file
6005	Element ’%s’ does not exist in definition file
6006	—
6007	Group ’%s’ does not exist in definition file
6008	—
6009	On reading attribute of group ’%s’
6010	No space left in data file for integer attribute of group ’%s’
6011	No space left in data file for real attribute of group ’%s’
6012	No space left in data file for string attribute of group ’%s’
6013	Group ’%s’ does not exist in data file
6014	On reading attribute of group ’%s’
6015	Integer attribute ’%s’ of group ’%s’not found
6016	No valid attribute name found
6017	Real attribute ’%s’ of group ’%s’not found
6018	No valid attribute name found
6019	String attribute ’%s’ of group ’%s’not found
6020	No valid attribute name found
-6021	No more data groups available in DefinitonData file ’%s’ No data groups available in DefinitonData file ’%s’
-6022	No more data groups available in data file ’%s’ No data groups available in data file ’%s’
-6023	No more elements available in DefinitionData file ’%s’ No elements available in DefinitionData file ’%s’
-6024	No more elements available in definition file ’%s’ No elements available in definition file ’%s’
-6025	No more cells available in DefinitionData file ’%s’ No cells available in DefinitionData file ’%s’
-6026	No more cells available in definition file ’%s’ No cells available in definition file ’%s’
-6027	No more defined groups available in DefinitionData file ’%s’ No defined groups available in DefinitionData file ’%s’
-6028	No more defined groups available in definition file ’%s’ No defined groups available in definition file ’%s’
6029	On reading first variable pointer table
6030	On reading variable pointer table, table %ld
6031	On reading variable pointer table, table %ld
6032	On reading variable pointer table, table %ld
6033	Maximum size reached in DataDefinition file
6034	Maximum size reached in data file
7001	No entry found in hash table for key ’%s’

Nr	Error/Warning description
7002	Unable to read next pointer (file write only?)
7003	During reading of next pointer
7004	During reading of cell structure
7005	During reading of element structure
7006	During reading of group structure
7007	During reading of data structure
8001	Data file ’%s’ has already been opened
8002	Definition file ’%s’ has already been opened
8003	Maximum number (=%d) of open files has been achieved
8004	Header was not written to data file ’%s’
8005	Hashtable was not written to data file ’%s’
8006	Hashtable was not written to data file ’%s’
8007	Unable to access data file ’%s’
8008	During reading header of data file ’%s’
8009	File ’%s’ is not a NEFIS data file
8010	During reading hashtable of data file ’%s’
8011	Unable to write header of definition file ’%s’
8012	Unable to write hashtable of definition file ’%s’
8013	Unable to write hashtable of definition file ’%s’
8014	Unable to access definition file ’%s’
8015	during reading header of definition file ’%s’
8016	File ’%s’ is not a NEFIS definition file
8017	On reading hashtable of definition file ’%s’
8018	Unable to write header of DefinitionData file ’%s’
8019	Unable to write hashtable of DefinitionData file ’%s’
8020	Unable to write hashtable of DefinitionData file ’%s’
8021	Unable to access DefinitionData file ’%s’
8022	during reading header of DefinitionData file ’%s’
8023	File ’%s’ is not a NEFIS DefinitionData file
8024	On reading hashtable of DefinitionData file ’%s’
8025	Unable to close data file ’%s’
8026	Unable to close definition file ’%s’
8027	Unable to close DefinitionData file ’%s’
8028	—
8029	File ’%s’ can not be opened with unsupported NEFIS access type ’%c’
8030	Cannot open file ’%s’ for access type ’%c’
8031	File ’%s’ can not be opened as read only
9001	The variable MAX_VAR_GROUPS needs to be increased. Contact Deltares | Delft Hydraulics
9002	Element "%s" of group "%s" not found on file "%s"
10001	This size of integer (%d) is not supported
10002	This size of real (%d) is not supported
10003	This size of character (!=1) is not supported
10004	This size of complex (%d) is not supported
10005	This size of logical (%d) is not supported
10006	This element type is not supported ’%s’

C Demonstration programs

Two demonstration programs are appended, DEMO_01 and DEMO_02. DEMO_01 particularly demonstrates how the stored and retrieved sequences can be handled. The program uses the file DEMO_01.INP and generates the definition file (DEMO_01.DEF) and the data file (DEMO_01.DAT). The program DEMO_01 is the example mentioned earlier (see Examples option A). The measurement data is defined as:

velocity
depth 1: 1000×location number+10×time+1
depth 2: 1000×location number+10×time+2
depth 3: 1000×location number+10×time+3

depth
1000×location number + 10×time + 4

Program DEMO_02 shows an example of how variable dimensions are handled and also demonstrates how easy it is to use only the relevant part of available data. This program does not need an input file.

C.1 Demonstration program 1

```
program demo01
    implicit none
!
!     Company name                     : Deltares | Delft Hydraulics
!                                        P.O.Box 177
!                                        2600 MH Delft
!                                        The Netherlands
!     DESCRIPTION :
!
!     This program demonstrates how the NEFIS functions can be used
!     to write data to a NEFIS file, and how this data can be
!     retrieved again from the same file.
!
!=======================================================================
!     ..
!     .. Local Scalars ..
!
    character*1024 errstr        ! character string to catch the nefis error message
    character coding*1           ! indicates y/n neutral representation of data
    integer :: dummy             ! dummy parameter in function calls
    integer :: error             ! contains return-value of nefis functions
    integer :: i                 ! loop control variabel, array index
    integer :: j                 ! loop control variabel, array index
    integer :: k                 ! loop control variabel, array index
    integer :: obsfil            ! unitnumber of user's observation file
!
!     .. local arrays ..
!
    character elmnms(2)*14       ! cell element names
    integer :: fds               ! nefis file descriptor
    integer :: grpdms(5)         ! dimensions of data group to define
    integer :: grpord(5)         ! order information of data group to define
    integer :: usrord(5)         ! ordering information of user data
    integer :: usrind(3,5)       ! ordering information of user data
    real    :: veloctiy(3,4)     ! array to contain velocity-element info of 4 points
    integer :: depth(12)         ! array to contain depths
    integer :: obsdat(4,100,10)  !                 .. array to contain observed data
!     ..
!     .. external functions ..
```

```
!
    integer :: crenef           ! nefis-function: open a data and definition file
    integer :: defelm           ! nefis-function: define an element
    integer :: defcel           ! nefis-function: define a cell
    integer :: defgrp           ! nefis-function: define a group
    integer :: credat           ! nefis-function: create space for data on data file
    integer :: putelt           ! nefis-function: write data of 1 or more elements
!
!     .. to data file
!
    integer :: getelt           ! nefis-function: retrieve data of 1 or more
!
!     .. elements from data file
!
    integer :: clsnef           ! nefis-function: close a data and definition file
!
!=======================================================================
!
      obsfil = 11
      coding = ’b’ ! let us write the data to a big endian format
      rdwr   = ’c’ ! create a nefis file set
!                 ..
      write(*,’(’’Demo_01: Open NEFIS data and definition file’’)’)
      error = crenef(fds, ’data_d01.def’, ’data_d01.def’, coding, rdwr)
      if (error /= 0) goto 9999
!=======================================================================
!
!    FIRST, LET’S DEFINE SOME ELEMENTS
!
!    DEFINE A 1 DIMENSIONAL ELEMENT OF 3 REALS,
!    NAMED: MEAN VELOCITY
!
    write(*,’(’’demo_01: Define ELEMENT: MEAN VELOCITY (3)’’)’)
    error = defelm (fds, ’MEAN VELOCITY’, ’REAL’, 4, &
                    ’VELOCITY’, ’[M/S]’, &
                    ’Mean velocity in centre of river at ’ // &
                    ’3 different levels’, 1, 3)
    IF (ERROR /= 0) GOTO 9999
!
!   DEFINE A (0 DIMENSIONAL) ELEMENT OF 1 REAL,
!   NAMED: WATERDEPTH
!
    write(*,’(’’demo_01: Define ELEMENT: WATERDEPTH’’)’)
    dummy = 1
    error = defelm (fds, ’WATERDEPTH’, ’REAL’, 4, ’DEPTH’, &
                   ’[M]’, ’DEPTH AT CENTRE OF RIVER’, 1, dummy)
    if (error /= 0) goto 9999
!
!   LET’S DEFINE A CELL TO CONTAIN OBSERVATIONS AT A
!   CERTAIN POINT AND A CERTAIN PLACE,
!   NAMED: OBSERVATION
!
    elmnms(1) = ’MEAN VELOCITY’
    elmnms(2) = ’WATERDEPTH’
    write(*,’(’’demo_01: Define CELL: ’’, &
              ’’OBSERVATION = MEAN VELOCITY + WATERDEPTH’’)’)
    error = defcel(fds, ’OBSERVATION’, 2, elmnms)
    if (error /= 0) goto 9999
!
!   DEFINE A GROUP FOR 10 DIFFERENT LOCATIONS,
!   ABLE TO CONTAIN 100 OBSERVATIONS (TIME SERIES)
!   FOR EACH LOCATION, NAMED: RIVER DATA
!
    grpdms(1) = 100 ! MAX. 100 OBSERVATIONS FOR EACH LOCATION
    grpdms(2) = 10  ! MAX. 10 LOCATIONS
    grpord(1) = 2
    grpord(2) = 1
!
!   CELLS WILL BE STORED IN THE FILE IN THE ORDER:
!   (1,1), (1,2) ..... (1,10), (2,1), (2,2).... ETC.
!
    write(*,’(’’demo_01: Define GROUP:’’, ’’ RIVERDATA = OBSERVATION (100,10)’’)’)
    error = defgrp (fds, ’RIVERDATA’, ’OBSERVATION’, 2, grpdms, grpord)
!
    if (error /= 0) goto 9999 ! END OF DEFINITION PART
!
!=======================================================================
!
!   NOW, LET’S CREATE SPACE ON THE DATA FILE FOR
!   RED RIVER DATA
!
    write(*,’(’’demo_01: Create space for data labelled: RED RIVER,’’, &
              ’’ using THE RIVERDATA GROUP DEFINITION’’)’)
    error = credat (fds, ’RED RIVER’, ’RIVERDATA’)
    if (error /= 0) goto 9999
!
!   NOW, READ ALL FIELD OBSERVATIONS FROM A FILE
!
    write(*,’(’’demo_01: Read observation data from input file’’, &
              ’’ (not a NEFIS action)’’)’)
!
    open (obsfil,file=’observ.inp’)
    do i = 1, 10
        read (obsfil,*)
        do j = 1, 100
            read (obsfil,*) (obsdat(k,j,i), k=1,4) ! VELOCITIES AND WATERDEPTH AT LOCATION I
        enddo
    enddo
    close (obsfil)
!=======================================================================
!
!   OBSERVATIONS CAN BE WRITTEN TO THE NEFIS DATA FILE
!   FOR EXAMPLE CELL AFTER CELL
!
    usrord(1) = 1
    usrord(2) = 2
!
!   THIS IS THE FORTRAN ORDER, IE.:
!            (1,1), (2,1) .. (100,1), (1,2), (2,2) .. ETC.
!
!     write(*,’(’’demo_01: Write DATA to NEFIS file, ONE cell at a time’’)’)
!     do 40 i = 1, 100
!        do 30 j = 1, 10
!           usrind(1,1) = i
!           usrind(2,1) = i
!           usrind(3,1) = 1
!           usrind(1,2) = j
!           usrind(2,2) = j
!           usrind(3,2) = 1
!           usrind(1,3) = 3
!           usrind(2,3) = 3
!           usrind(3,3) = 3
!           usrind(1,4) = 4
!           usrind(2,4) = 4
!           usrind(3,4) = 4
!           usrind(1,5) = 5
!           usrind(2,5) = 5
!           usrind(3,5) = 5
!           write(*,’(’’demo_01: Data: ’’,i8)’) OBSDAT(4,I,J)
!           error = putelt (fds, ’RED RIVER’, ’WATERDEPTH’, usrind, usrord, obsdat(4,i,j))
!           if (error /= 0) goto 9999
!        enddo
!     enddo
!     write(*,’(’’demo_01: RED RIVER written in [sec]’’,1PE13.5)’) cpu2-cpu1
!
!   OR ALL CELLS TOGETHER (10*100 CELLS)
!
    usrind(1,1) = 1
    usrind(2,1) = 100
    usrind(3,1) = 1
    usrind(1,2) = 1
    usrind(2,2) = 10
    usrind(3,2) = 1
!
!   INDEX OF FIRST CELL TO STORE INFORMATION TO
!
    usrord(1) = 1
    usrord(2) = 2
!
!                 .. THIS IS THE FORTRAN ORDER, IE.:
!            (1,1), (2,1) .. (100,1), (1,2), (2,2) .. ETC.
!
    write(*,’(’’demo_01: Write the DATA, all cells at ONE go’’)’)
    error = putelt (fds, ’RED RIVER’, ’*’, usrind, usrord, obsdat)
    IF (ERROR /= 0) GOTO 9999
!
!   ALL DATA IS NOW STORED ON THE NEFIS DATA FILE
!
!=======================================================================
!
!   LET’S DO SOME RETRIEVAL
!
!   LET’S RETRIEVE THE VELOCITIES FROM LOCATIONS
!   6-9 AT TIME 54, IE. FROM
!   CELLS (54,6), (54,7), (54,8) AND (54,9).
!   THE PERFORMANCE WILL BE RATHER GOOD, BECAUSE THE
!   DATA ON THE NEFIS DATA FILE IS WRITTEN IN THIS
!   ORDER (SEE DEFGRP).
!
    usrind(1,1) = 6
    usrind(2,1) = 9
    usrind(3,1) = 1
    usrind(1,2) = 54
    usrind(2,2) = 54
    usrind(3,2) = 1
!
    usrord(1) = 2
    usrord(2) = 1
!
!   MEANS: FROM CELL (54,6), (54,7), (54,8) AND (54,9)
!
    write(*,’(’’demo_01: Start retrieval’’)’)
    Error = getelt (fds, ’RED RIVER’,’MEAN VELOCITY’, &
                    usrind, usrord, 48, velocity)
    if (error /= 0) goto 9999
!                 ..
    write(*,’(’’demo_01: Velocities at time 54’’)’)
    do i = 1, 4
        write (*,’(a,i2,’’:’’,3f8.1)’) ’  Location ’, i+5, (velocity(j,i), j=1,3)
    enddo
!
!   NOW, RETRIEVE AT LOCATION 7 THE WATERDEPTHS FROM
!   TIME 35-46
!
    usrind(1,1) = 7
    usrind(2,1) = 7
    usrind(3,1) = 1
    usrind(1,2) = 35
    usrind(2,2) = 46
    usrind(3,2) = 1
!
    usrord(1) = 2
    usrord(2) = 1
!
!   MEANS: FROM CELL (35,7), (36,7), (37,7) .... (46,7)
!
    error = getelt (fds, ’RED RIVER’, ’WATERDEPTH’, &
                    usrind, usrord, 48, depth)
    if (error /= 0) goto 9999
!                  ..
    write (*,’(’’demo_01: Waterdepths at location 7’’)’)
    do  i = 1, 12
        write (*,’(a,i2,’’:’’,f8.1)’) ’  Time ’, i+34, depth(i)
    enddo
!=======================================================================
!
!   close the NEFIS files
!
    write(*,’(’’demo_01: Close the NEFIS files’’)’)
    error = clsnef (fds)
!
9999 continue
!
    error = neferr( 0, errstr)
    write(*,’(a)’) trim(errstr)
    write(*,*)
    write(*,’(’’demo_01: End of demonstration’’)’)
end program
```

C.2 Demonstration program 2

```
program demo_02
    implicit none
!
!   Company name                      : Deltares | Delft Hydraulics
!                                       P.O.Box 177
!                                       2600 MH Delft
!                                       The Netherlands
!------------------------------------------------------------------------------
!   System: NEFIS
!
!   $Header: /delft3d/libraries/nefis/demo/demo_02/demo_02.f 2     10/03/06 9:56 Mooiman $
!------------------------------------------------------------------------------
!   Programmer                        : A. Hoekstra
!   Project                           : NEutral FIle Structure
!------------------------------------------------------------------------------
!     * * * * * * * * * * * * * DESCRIPTION * * * * * * * * * * * * *
!
!   - This demo-program demonstrates the use of NEFIS store-
!     and retrieval functions. Special is the use of a
!     datagroup with a variable dimension.
!
!     This program performs the following tasks:
!     - create an element, cel and a 3-d group defintion
!     - create a data group
!     - store data in this group
!     - retrieve data from this group, using a
!       different view
!     - retrieve data using a filter
!
!   Note: the error-return code from the NEFIS-functions is
!         not checked
!------------------------------------------------------------------------------
```

```
!
!    Scalars
!
    character*1024 errstr    ! character string to catch the NEFIS error message
    character coding*1       ! indicates Y/N neutral representation of data
    character rdwr*1         ! indicates read write acces of the file
    integer   error          ! contains return-value of NEFIS-functions
!
!   Arrays
!
    integer   fds            ! nefis file descriptor
!
!   Declarations of NEFIS-functions
!
    integer :: clsnef
    integer :: credat
    integer :: flsdat
    integer :: crenef
    integer :: neferr
!
!   Executable statements
!
!
!   Open a definition file
!
    coding = ’L’
    rdwr   = ’c’
    error = crenef (fds, ’data_d02.daf’, ’data_d02.daf’, coding, rdwr)
    if (error /= 0) goto 9999
!
!   Define element, cel, and group-definition
!
    call define (fds)
!
!   Create space for data
!
    error = credat (fds, ’GrpNaam’, ’Groep’)
    if (error /= 0) goto 9999
!
    error = flsdat (fds)
    if (error /= 0) goto 9999
!
!   Write data to file
!
    call putdat (fds)
!
!   Retrieve data, using a different view
!
    call dtview (fds)
!
!   Retrieve a part of the data
!
    call filter (fds)
!
!   Close the files
!
 9999 continue
!
    if (error.eq.0) error = clsnef (fds)
!
    error = neferr( 1, errstr)
!
end program
!==================================================================================
subroutine define (fds)
    implicit none
```

```
!
    integer   fds
!
    integer       error
    character*134 errstr
!
    integer :: grpdms(5)
    integer :: grpord(5)
!
    integer :: defcel
    integer :: defelm
    integer :: defgrp
    integer :: flsdef
    integer :: neferr
!
!   Executable statements
!
!   Define a simple element, type Real*4
!
    error = defelm (fds, ’ElmName’, ’Integer’,  4, &
                    ’ElmQuantity’, ’ElmUnity’, ’ElmDescription’, &
                    1, 1)
    if (error /= 0) goto 9999
!
!   Define a cel with only one real value
!
    error = defcel (fds, ’Cell’, 1, ’ElmName’)
    if (error /= 0) goto 9999
!
!   Define a 3-d group of dimension (3,5,0),
!   so a group with a variable dimension
!
    grpdms(1) = 3
    grpdms(2) = 5
    grpdms(3) = 0
    grpord(1) = 1
    grpord(2) = 3
    grpord(3) = 2
    error = defgrp (fds, ’Groep’, ’Cell’, 3, grpdms, grpord)
    if (error /= 0) goto 9999
!
!   Flush buffers to file
!
    error = flsdef (fds)
    if (error /= 0) goto 9999
!
 9999 continue
    error = neferr( 1, errstr)
end subroutine
!================================================================================
subroutine putdat (fds)
    implicit none
!
    character*1024 errstr
!
    integer   fds
!
    integer   start, stop, incr
    parameter (start=1, stop=2, incr=3)
    equivalence(aarray,array)
!
    character :: space*7
    integer   :: col
    integer   :: error
    integer   :: plane
    integer   :: row
    integer   :: uindex(3,5)
    integer   :: usrord(5)
    integer   :: array (3,5,7)
    integer   :: aarray (105)
!
    integer   :: flsdat
    integer   :: putelt
    integer   :: neferr
!
!   Executable statements
!
    space = ’       ’
!
!   Set view to (3,5,*)
!
    usrord (1) = 1
    usrord (2) = 2
    usrord (3) = 3
!
!   Define indices for each dimension
!
    uindex (start,1) = 1
    uindex (stop ,1) = 3
    uindex (incr ,1) = 1
    uindex (start,2) = 1
    uindex (stop ,2) = 5
    uindex (incr ,2) = 1
    uindex (start,3) = 1
    uindex (stop ,3) = 7
    uindex (incr ,3) = 1
!
!   Fill array with values
!
    do  plane = 1,7
        do  col = 1,5
            do  row = 1,3
                array (row, col, plane) = row*1000+col*100+plane
            enddo
        enddo
    enddo
!
!   Write data to file
!
    error = putelt (fds, ’GrpNaam’, ’*’ ,uindex, usrord, array)
    if (error /= 0) goto 9999
!
!   Flush the buffers
!
    error = flsdat (fds)
    if (error /= 0) goto 9999
!
!   Output data to screen
!
    write(*,’(’’ ARRAY(3,5,7) written to file:’’)’)
    do plane = 1,7
        do col = 1,5
            write (*,’(  3i10)’) (array(row,col,plane),row=1,3)
        enddo
        write (*,*)
    enddo
!
 9999 continue
    error = neferr( 1, errstr)
end subroutine
!==============================================================================
subroutine dtview (fds)
```

```
    implicit none
!
    character*1024 errstr
!
    integer   :: fds
!
    integer   :: start, stop, incr
    parameter (start=1, stop=2, incr=3)
!
    character :: space*7
    integer   :: col
    integer   :: error
    integer   :: plane
    integer   :: row
    integer   :: uindex(3,5)
    integer   :: usrord(3)
    integer   :: array (7,3,5)
!
    integer   :: getelt
    integer   :: neferr
!
!   Executable statements
!
    space = ’       ’
!
!   Change view to (*,3,5)
!
    usrord (1) = 3
    usrord (2) = 1
    usrord (3) = 2
!
!   define indices for each dimension
!
    uindex (start,1) = 1
    uindex (stop ,1) = 7
    uindex (incr ,1) = 1
    uindex (start,2) = 1
    uindex (stop ,2) = 3
    uindex (incr ,2) = 1
    uindex (start,3) = 1
    uindex (stop ,3) = 5
    uindex (incr ,3) = 1
!
!   Retrieve data
!
    error = getelt (fds, ’GrpNaam’, ’*’ ,uindex, usrord, 7*3*5*4, array)
    if (error /= 0) goto 9999
!
!   Output data to screen
!
    write(*,’(’’ Same values now retrieved in ARRAY(7,3,5)’’)’)
    do plane = 1,5
        do col = 1,3
            write (*,’(  7i10  )’) (array(row,col,plane),row=1,7)
        enddo
        write (*,*)
    enddo
!
 9999 continue
    error = neferr( 1, errstr)
end subroutine
!==================================================================================
subroutine filter (fds)
    implicit none
!
    character*1024 errstr
```

```
!
    integer   :: fds
!
    integer   :: start, stop, incr
    parameter (start=1, stop=2, incr=3)
!
    character :: space*7
    integer   :: col
    integer   :: error
    integer   :: plane
    integer   :: row
    integer   :: uindex(3,5)
    integer   :: usrord(3)
    integer   :: array (4,2,3)
!
    integer   :: getelt
    integer   :: neferr
!
!   Executable statements
!
    space = ’       ’
!
!   Change view to (*,3,5)
!
    usrord (1) = 3
    usrord (2) = 1
    usrord (3) = 2
!
!   Define indices and step for each dimension
!   The stepsize of 2 creates a filter
!
    uindex (start,1) = 1
    uindex (stop ,1) = 7
    uindex (incr ,1) = 2
    uindex (start,2) = 1
    uindex (stop ,2) = 3
    uindex (incr ,2) = 2
    uindex (start,3) = 1
    uindex (stop ,3) = 5
    uindex (incr ,3) = 2
!
!   Retrieve data
!
    error = getelt (fds, ’GrpNaam’, ’*’, uindex, usrord, 4*2*3*4, array)
    if (error /= 0) goto 9999
!
!   Output data to screen
!
    write(*,’(’’ Every other value retrieved in ARRAY(4,2,3)’’)’)
    do plane = 1,3
        do col = 1,2
            write (*,’(  4i10  )’) (array(row,col,plane),row=1,4)
        enddo
        write (*,*)
    enddo
!
 9999 continue
    error = neferr( 0, errstr)
    write(*,’(a)’) trim(errstr)
end subroutine
```

D Release notes

D.1 Differences between Version 5.00 and 4.00

NEFIS5 support file sizes up to $2^{64} - 1$ byte. The internal pointers to the data on the file is 64-bits, these pointers are also written to the NEFIS-file. As a consequence the files made by NEFIS5 and NEFIS4 are not compatible. The Application Programming Interface (API) is compatible with NEFIS4. NEFIS5 will read NEFIS4 data, and if necessary NEFIS5 will write 32-bit data to an existing NEFIS4 file. New files will always be a NEFIS5 file.

D.2 Differences between Version 4.00 and 3.10

D.2.1 Improvements

The following improvements are obtained:

Access	For reading a NEFIS files no write access is needed.
One file	If the same name for the data and the definition file is used, one NEFIS file is created. That NEFIS files has the same performance and usability as when using two files.
Variable dimension	The performance to retrieve data from a group with variable dimension is highly improved.
Elements	The performance for cells containing more than 40 elements is highly improved.
Qdblok3	The object qdblok3 is not needed anymore. This object contained a separated data block (Fortran: Data Block), which was separately compiled. The data is now integrated into the NEFIS library.

D.2.2 New functions

The following function are added to NEFIS version 4.00:

Name	**Definition**
Crenef	Open or create a NEFIS file set at once, you have to supply the access type of the files. So, for post-processing the "read" permission is enough to make graphs.
Clsnef	Close the NEFIS file set. Depending on the access type the buffer will be written to file or not.
Getels	For all architectures the function Getels is added. This function reads one or all string elements from the data file.
Neferr	NEFIS error function to retrieve more sensible error messages.
Putels	For all architectures the function Putels is added. This function writes one or more character elements to the data file
Inqfcl/Inqncl	Reading first and next cell definition from the definition file
Inqfel/Inqnel	Reading first and next element definition from the definition file
Inqfgr/Inqngr	Reading first and next group definition from the definition file

D.2.3 Outdated functions

The next functions are still available in NEFIS but are not supported anymore:

Name	Definition
Cldfnf	Close the definition file without flushing the buffer (timestamp does not change due to this function).
Cldtnf	Close the data file without flushing the buffer (timestamp does not change due to this function).
Clsdat	Flush buffers to the data file and close that file.
Clsdef	Flush buffers to the definition file and close that file.
Opndat	Create or open a NEFIS data file.
Opndef	Create or open a NEFIS definition file.

D.2.4 Unavailable functions

The following functions are not available anymore within this version NEFIS.

Name	Definition
Getelm	Replaced by function Getelt.
Putelm	Replaced by function Putelt.

D.3 Differences between Version 3.10 and 3.00

D.3.1 Improvements

Two functions are added to the library, which avoid changing the time stamp of the NEFIS files. These functions can only be used when reading data from file, if also data is written to file then the function Clsdat and Clsdef have to be used.

D.3.2 New functions

The following functions are added to NEFIS version 3.10:

Name	Definition
Cldfnf	Does NOT write buffers to the definition file and close the file (consequence: timestamp does not change).
Cldtnf	Does NOT write buffers to the definition file and close the file (consequence: timestamp does not change).

D.4 Differences between Version 3.00 and 2.00

D.4.1 Improvements

Performance improvement for reading and writing data to the files when using the functions PUTELT and GETELT. The improvement depends on the chosen structure. Depending on the structure the performance reach from several percents to factors (upto factor 13 in some tests).

D.4.2 New functions

The following functions are added to NEFIS version 3.00:

Name	Definition
Getelt	Replaces the GETELM function. GETELT outperforms GETELM which is now outdated. The GETELM and GETELT functions can be used interchangeable for data groups without variable dimension.
Gethdf	This function gives information from the header of a NEFIS definition file.
Gethdt	Give information from the header of a NEFIS data file.
Inqfia	Give the first name and value of an integer attribute of a data group.
Inqfra	Like INQFIA, but here for the real attributes.
Inqfsa	Like INQFIA, but here for the real attributes.
Inqmxi	Give the maximum used index of the free dimension of a data group.
Inqnia	This function gives the next name and value of an integer attribute of a data group (see INQFIA function).
Inqnra	Like INQNIA, but here for the real attributes.
Inqnsa	Like INQNIA, but here for the real attributes.
Putelt	Outperforms PUTELM and replaces it. As a result, PUTELM is considered to be outdated. However, PUTELM and PUTELT can be used interchangeable for data groups without variable dimension.

D.4.3 Outdated functions

The next functions are still available in NEFIS but are not supported anymore:

Name	Definition
Getelm	Replaced by function Getelt.
Putelm	Replaced by function Putelt.

Deltares systems

PO Box 177
2600 MH Delft
Rotterdamseweg 185
2629 HD Delft
The Netherlands

+31 (0)88 335 81 88
sales@deltaressystems.nl
www.deltaressystems.nl