ERF-HDF5 Specification
Version 2.2 Rev.1

ESI Software

November 2016
ESI Group

Contents

1 Introduction

ERF-HDF5 is the new ESI Result database file standard. It is based on the file standard HDF5 (The HDF group, http://hdfgroup.com/HDF5).
ERF-HDF5 is an open data format to store, access, manage and exchange simulation data. ERF-HDF5 data files are portable across different computing platforms and architectures. The ERF data model is based on a simple but versatile block structure and is therefore easily extensible to meet future requirements. The ERF-HDF5 specification is freely distributed.

2 Data Block Structure

2.1 Overview

The data in ERF files is stored in ERF-Blocks. There are different types of ERF-Blocks to store different kind of data. Generally, the ERF-Blocks can be divided into three categories:

- constant data blocks,
- single-state blocks and
- multi-state blocks.

While the constant data blocks contain constant data like connectivities, variable declarations or unit definitions, the single-state and the multi-state blocks contain changeable, e.g. time-dependent, data.

Multi-state blocks contain extendable datasets (see HDF5 documentation) to store data of multiple states (results, coordinates, etc.) compactly in a single array. These extendable datasets have – unlike the fix-dimensioned single-state block datasets – one unlimited dimension. The main purpose of multi-state blocks is to store time-series data (e.g. of thousands of states).

The simulation state is defined by an n-dimensional index (e.g. n=2: "*Time*", "*Loadcase*"). ERF-blocks with no index (n=0) are considered as constant data blocks. Constant data blocks provide global data that is valid for the whole model and applicable at all simulation states.

Table: Block types and properties

block	Function *erf{ _read_ | _write_ }*	mandat. optional	c = constant s = single-s. m = multi-s.	index mandat. optional	Refer to variable groups
ERF Header Block	*fileheader*	m	c	n/a	no
Block 10 – System Block	*system*	m	c	n/a	no
Block 20 – Index Definition Block	*indices*	m	c	n/a	yes
Block 30 – Variable Definition Block	*variable*	m	c	n/a	no
Block 32 – Variable Group Definition Block	*variablegroup*	m	c	n/a	no
Block 40 – Text Block	*text*	o	c	n/a	no
Block 100 – Parts Block	*parts*	o	c | s	o	no
Block 200 – Attributes Block	*attributes*	o	c | s	o	no
Block 250 – Identifiers Block	*identifiers*	o	c | s	o	no
Block 300 – Element Connectivity Block	*connectivities*	o	c | s	o	yes
Block 400 – Collector Block	*collector*	o	c | s	o	no
Block 500 – Properties Block	*properties*	o	c | s	o	no
Block 10xx – Entity Result Block	*entityresults*	o	c | s	o	yes
Block 1050 – Multi-State Entity Result Block for Real Numbers	*multientityresults*	o	m	m	yes
Block 11xx – Intra-Elemental Result Block	*elementresults*	o	c | s	o	yes
Block 1150 – Multi-State Intra-Elemental Result Block for Real Numbers	*multielementresults*	o	m	m	yes
Block 2100 - Activation Flag Block	*activflags*	o	c | s	o	no
Block 2150 – Multi-State Activation Flag Block	*multiactivflags*	o	m	m	no
Block 10xxx – Matrix Block	*{dense|triangular|sparse}matrix*	o	c | s	o	yes

2.2 Internal File Structure

HDF5 organizes the file data in groups, datasets and attributes. An HDF5 group is analogous to a file system directory and a HDF5 dataset to a file. HDF5 attributes are small meta-data objects, which can be attached to groups or datasets. In the ERF-HDF5 file huge data are stored in datasets, whereas small data, describing the intended usage of the datasets, are stored as attributes.

The figure below shows the hierarchical storage of data objects in an ERF-HDF5 file.

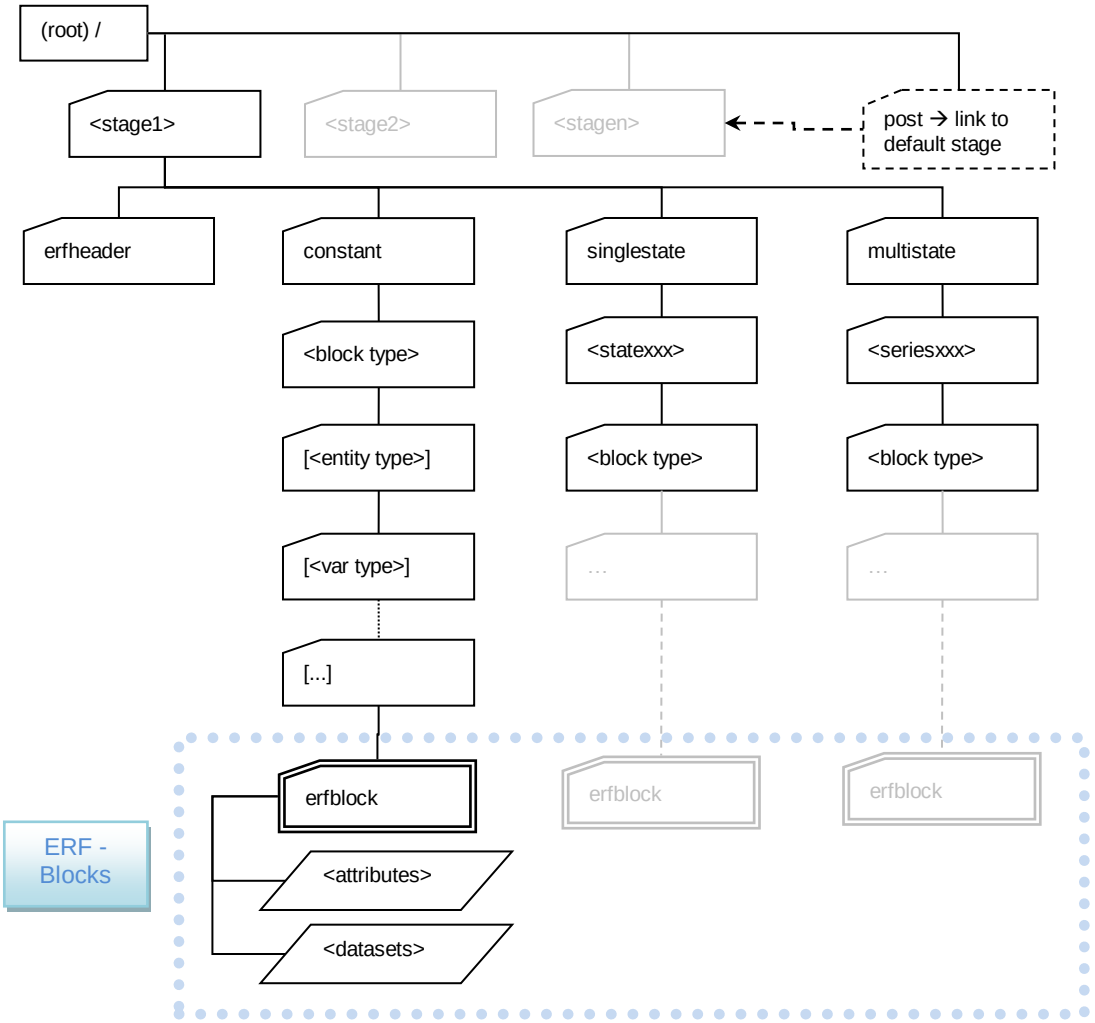


Figure: ERF-HDF5 file storage scheme

This hierarchical storage scheme is specific to the application. The figure below shows a sample file (in the HDF file explorer *HDFView*) written by *PAM-CRASH* solver.

Since all the descriptive data is stored in the ERF-blocks, the post-processor can identify and find all ERF-blocks by iterating through the file (e.g. by using the HDF5 function *H5Literate*).

ERF-HDF5 datasets are referenced by names (see column “Variable” in the block descriptions). In addition to the name of the dataset, HDF5 requires to specify a data-space of a particular type. Currently two types of data-spaces are used, *H5S_SCALAR* for scalar variables and *H5S_SIMPLE* for arrays. In the ERF block descriptions arrays are declared by the use of one or multiple indices in square brackets (C syntax), e.g. `x[i][j][k]` for an array of rank 3.

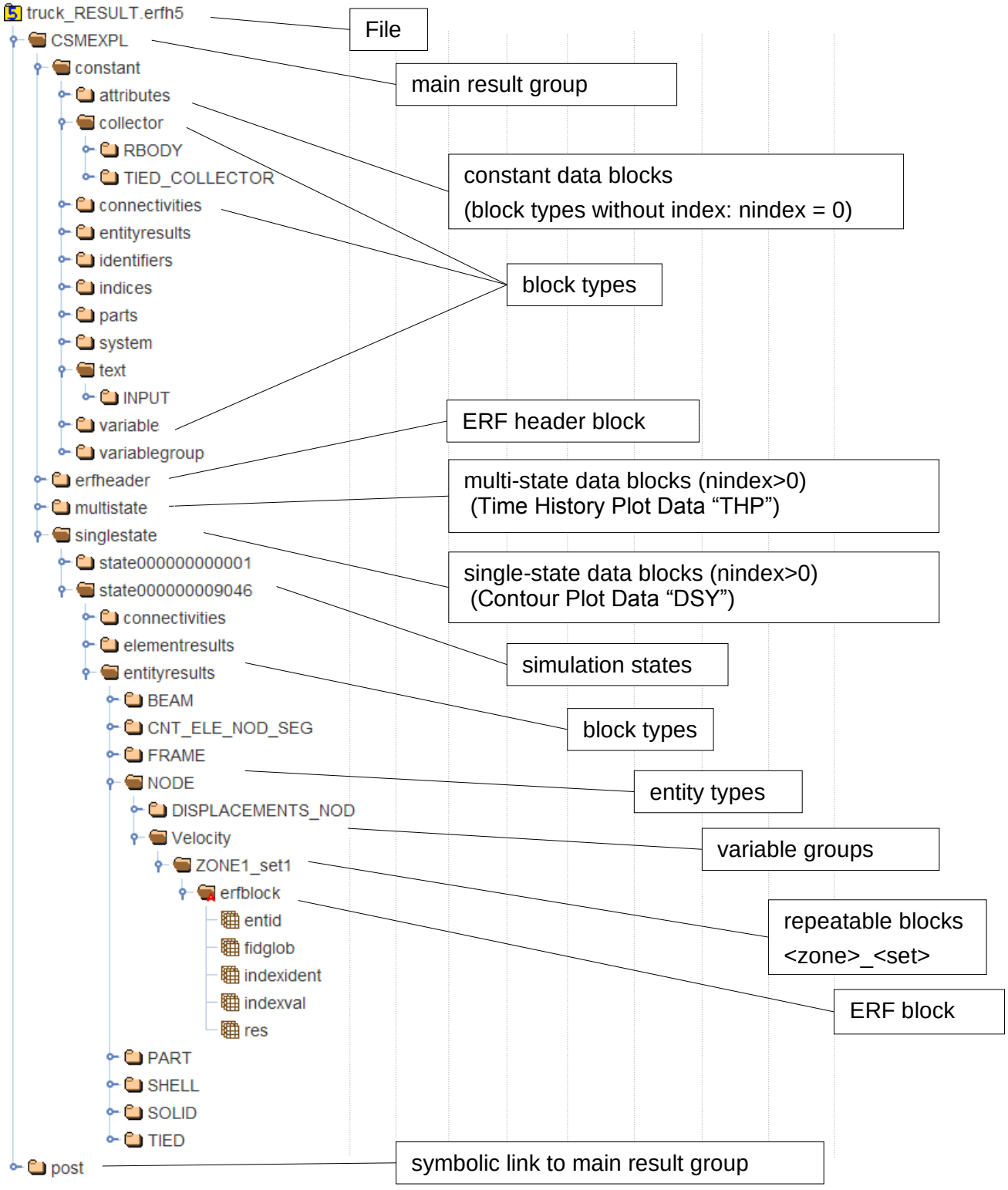


Figure: HDFView snapshot

2.3 Data Types and Formats

Generally, there are 3 kinds of data, floating point, integer and character data. In the block descriptions they are specified as follows:

Type	Kind of data	Declaration in C	Fortran	Examples
{A¦D}:INT	integer	int	INTEGER*4	values, numbers, flags
{A¦D}:INTID	integer	int long long int	INTEGER*4 INTEGER*8	entity identifiers
{A¦D}:LONG	integer	long long int	INTEGER*8	array pointers, offsets, counts
{A¦D}:FLOAT	floating point	float double	REAL*4 REAL*8	results, coefficients, coordinates
{A¦D}:CHAR[n]	n characters	char x[n]	CHARACTER(n)	types, names, expressions

In HDF5 data can be stored as an *Attribute* or as a *Dataset*, indicated by the prefix “A:” or “D:”.

Note that the precision and size of floating point data is not prescribed in the specification. However, I/O software has to make sure that datasets are read or written with the appropriate format. The HDF library provides functionality to define and check the data-type and size of data. HDF also allows for automatic data conversions.

In order to facilitate the handling of strings, the character variables are stored in character arrays of a fixed length. End-of-string characters (like ‘\0’ in C) are not supported. The length of character variables is given in the specification (value in squared brackets “CHAR[n]”). Character variables must be space-padded (see example below). The default length of character strings is n=ESL:

ERF Default String Length	CHAR[ESL] with ESL = 256	for all “etyp” parameters

Character variables may be any string of ASCII characters not containing a slash or a dot (‘/’ and ‘.’, which are reserved as HDF path separators) and starting with a non-space character. However, users are advised to avoid the use of punctuation and non-printing characters, as they may create problems for other software.

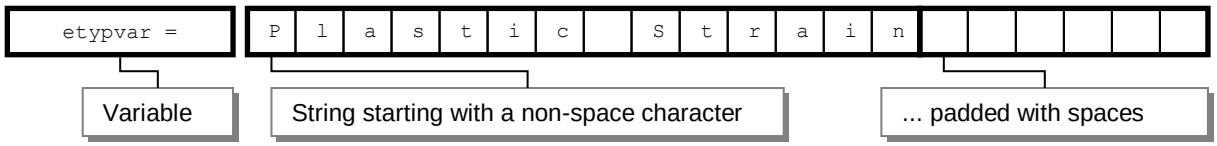


Figure: Example of a character variable

2.4 Frame Definitions

Frames are local or global Cartesian or non-Cartesian coordinate systems. Frames can be referred by any result variable stored in ERF element- or entityresults blocks. The frames are stored in entityresults blocks. A Cartesian frame could be written, for instance, as a 3x3 matrix containing the vector basis (see figure below).

$$\vec{\vec{\sigma}} = \sigma_{ij}\vec{e}_i\vec{e}_j$$

variable coordinates frame

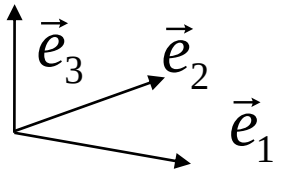


Figure: Variable in the Cartesian frame

The frame entity type is defined by the parameter *etypframe*. Any result value can either refer to the frame ID stored in the optional array *fid* or – if *fid* does not exist (*fswitch* = 0) - to the frame ID *fidglob* (see table below).

Table: Frame references in ERF result blocks

Frame type	etypframe	A:CHAR[ESL]	e.g. FRAME, ELEMENT_FRAME
Frame control switch	fswitch = { 0 | 1 | 2}	A:INT	= 0 → use fidglob = 1 → single fid per element = 2 → fid per intra-location

Option: frame IDs	if(fswitch == 0)fidglob
	
	if(fswitch == 1)
	(i=0; i<nent; i++) fid[i]
	
	if(fswitch == 2)
	(i=0; i<nele; i++)
	(j=0; j<numl; j++)
	fid[i][j]
		D:INTID	= 0 → global cartesian
			> 0 → frame id
			< 0 → local element frame

2.5 Distributed Memory Processing (DMP)

In order to do reduction operations for merging the domain files of parallized DMP solvers, reduction operators and weights can be stored in the result blocks (see table below). Note that these operations require global and unique entity IDs over all compute domains.

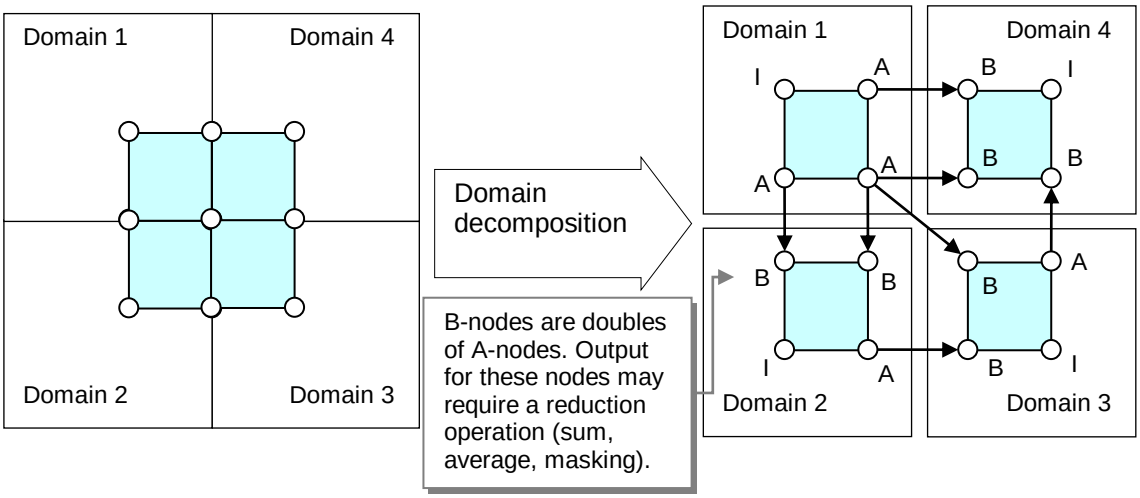


Figure: DMP domain decomposition scheme of *PAM-CRASH* for Finite Elements

Table: DMP reduction operators and weights

DMP reduction switch	dmpswitch = { 0 | 1 }	A:INT	0: off, 1: on
Option: DMP reduction operators	if(dmpswitch == 1) (i=0; i<ncoo; i++) operator[i]	D:CHAR[ESL]	OR, MAX, MIN, SUM, MEAN
Option: DMP domain weights	if(dmpswitch == 1) (i=0; i<nent; i++) weight[i]	D:FLOAT	

3 ERF Block Definitions

3.1 Simulation State Index

The ERF distinguishes between constant and variable data. Every variable data block is tagged with an index. The index must be declared with the index definition block (ERF-block type 20).

The index can be single- or multidimensional. The most common single-dimensional index is the progression parameter "*Time"*. As a two-dimensional index one may find "Loadcase" and "*Time"*.

The index data is stored in two arrays: *indexval* (type FLOAT) and *indexident* (type INT). The values of both arrays are bijective. The example below shows how these arrays may be filled.

state	indexident[0]	indexval[0]	indexident[1]	indexval[1]
	"Loadcase"	"Force"	"Timestep"	"Time"
1	1	1.0	1	0.0
2	1	1.0	1224	10.0
3	1	1.0	3486	20.0
4	2	-1.0	1	0.0
5	2	-1.0	1224	10.0
6	2	-1.0	3486	20.0

Figure: Example of a two-dimensional index ("*Loadcase"*, "*Time"*).

From ERF version 2.0 on data can be stored incrementally. The new parameter *indexic* flags ERF blocks containing incremental data. In order to reconstruct the full data array of any state, one has to go through all previous incremental states starting from the last full state. The figure below illustrates that. Note that this new option is only available for single-state - entityresults, intra-elemental results,, attributes and activation flag blocks (block types 200, 100x, 110x, 2100).

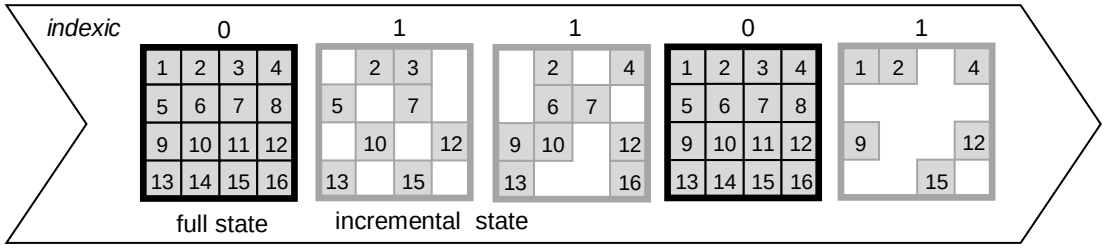


Figure: Storage of Incremental data

The following tables show the definition of the index in single- and multistate blocks.

Single-State Index			
No of indices	nindex	A:INT	
Incremental state flag	indexic	A:INT	indexic=0: full data block indexic=1: incremental block
Index identity	(i=0; i<nindex; i++) indexident[i]	D:INT	e.g. solution increment
Index Value	(i=0; i<nindex; i++) indexval[i]	D:FLOAT	e.g. time,freq,force, …

Multi-State Index			
No of indices	nindex	A:INT	
No of states	nstate	A:INT	
Index identity	(i=0; i<nstate; i++) (j=0; j<nindex; j++) indexident[i][j]	D:INT	e.g. solution increment
Index Value	(i=0; i<nstate; i++) (j=0; j<nindex; j++) indexval[i][j]	D:FLOAT	e.g. time,freq,force, …

3.2 ERF Header Block

Name	Variable [=value]	type	comments
ERF header	erfheader = < 8 characters signature: '\211' 'E' 'R' 'F' '\r' '\n' '\032' '\n' > < 32 characters for version numbers: major minor release > < 24 characters unused >	D:CHAR[64]	signature (8 x ASCIIC): Hex: 89 45 52 46 0d 0a 1a Dec: 137 69 82 70 13 10 26 10 version: 3 integer values separated by spaces: major minor release

Remarks:

Purpose of the header is to identify the ERF-HDF5 file format and to check the format version for compatibility.

The version is defined by three numbers separated by spaces and stored in the character field erfheader[8] … erfheader[39]. The three version numbers are to be interpreted as follows:

1) major version (one-based) – for major specification changes;
2) minor version (zero-based) – for minor specification changes;
3) release version (zero-based) – for software changes.

Example: major = 2, minor = 2, release = 0 → '2 2 0'

3.3 Block 10 – System Block

Name	Variable [= value]	type	comments
Block key	block = 10	A:INT	
Block type	blocktype="system"	A:CHAR[ESL]	
Model title	title	A:CHAR[ESL]	
System	sys	A:CHAR[ESL]	
Solver name	solver name	A:CHAR[ESL]	
Solver version	solver vers	A:CHAR[ESL]	
Creation date	cdate	A:CHAR[8]	YYYYMMDD
Creation time	ctime	A:CHAR[8]	HHMMSS
Modification date	mdate	A:CHAR[8]	YYYYMMDD
Modification time	mtime	A:CHAR[8]	HHMMSS
No of base units	nbunit	A:INT	see table of 7 SI base units
No of derived units	ndunit	A:INT	
Length udbid, udbexp	lenudb	A:INT	$\text{lenudb} = \sum_{\text{ndunit}} \text{ndbunit[i]}$
Base unit IDs	(i=0; i<nbunit; i++)ubid[i]	D:INT	Used to specify derived units
Base unit type	(i=0; i<nbunit; i++)ubtyp[i]	D:CHAR[ESL]	- Predef. values, see table
Base unit name	(i=0; i<nbunit; i++)ubnam[i]	D:CHAR[ESL]	e.g. mm, hour, etc.
Shift val to SI unit	(i=0; i<nbunit; i++)ubshift[i]	D:FLOAT	allows for unit conversions
Factor to SI unit	(i=0; i<nbunit; i++)ubcon[i]	D:FLOAT	allows for unit conversions
Derived unit IDs	(i=0; i<ndunit; i++)udid[i]	D:INT	Used to specify data units
Derived unit name	(i=0; i<ndunit; i++)udnam[i]	D:CHAR[ESL]	e.g. MPa, kN, J
Derived unit scale f.	(i=0; i<ndunit; i++)udscal[i]	D:FLOAT	e.g. 10^6
No of base units to specify derived units	(i=0; i<ndunit; i++)ndbunit[i]	D:INT	
Base unit IDs to build derived units	(i=0,k=0; i<ndunit; i++) (j=0; j<ndbunit[i]; j++) udbid[k++]	D:INT	Defined above
Base unit exponents	(i=0,k=0; i<ndunit; i++) (j=0; j<ndbunit[i]; j++) udbexp[k++]	D:FLOAT	e.g. Mpa = 10^6kg/s²/m allows for fractional units (fractals)

Notes to Data Units:

- All physical data may have associated units;
- Physical data without associated units treated as dimensionless;
- Units specified by multiplicative series as indicated below:

$$U = \prod_k V_{basek}^{\lambda_k} \prod_l V_{derived\ l}^{\mu_l}$$

with:

U	data unit
V_i	Component unit (base or derived)
λ_i , μ_i	component exponents

The base and derived unit components are to be defined in the system block as follows:

$$V_{basei} = \alpha_{basei}(B_i - S_i)$$

$$V_{derived\ j} = \alpha_{derived\ j} \prod_i V_{basei}^{v_i}$$

with:

B_i	SI standard unit
S_i	shift value to SI standard unit
$\alpha_{base\ i}$	conversion factor to SI standard unit
$\alpha_{derived\ j}$	unit scale factor of derived unit
v_i	exponent for the base unit to build the derived unit

Note: λ_i , μ_i, $v_i \neq 1$ only allowed for non-shifted base units with $S_i = 0$

In the table below there are the 7 predefined SI base units. However, the number of base units can be of arbitrary size.

Table: Predefined base unit types

UBTYP	SI Standard Unit	Unit symbol
Length	Metre	m
Mass	Kilogram	kg
Time	Second	s
Electric Current	Ampere	A
Thermodynamic Temperature	Kelvin	K
Amount of Substance	Mole	mol
Luminous Intensity	Candela	cd

3.4 Block 20 – Index Definition Block

Name	Variable [= value]	type	comments
Block key	block = 20	A:INT	
Block type	blocktype = "indices"	A:CHAR[ESL]	
Number of Indices	nindex	A:INT	nindex > 0
variable group to declare the index	etypvar	A:CHAR[ESL]	ncoo(etypvar) = nindex

3.5 Block 30 – Variable Definition Block

Name	Variable [= value]	type	comments
Block key	block = 30	A:INT	
Block type	blocktype="variable"	A:CHAR[ESL]	
Variable identifier	varkey	A:CHAR[ESL]	e.g. "VPS:Thickness"
Variable class	varclass	A:CHAR[ESL]	"CLASS STRESS", …
Variable name	varnam	A:CHAR[ESL]	"MembraneStrain", …
Independent coordinates	ncoo	A:INT	
Variable rank	rank	A:INT	matrix, vector, tensor: >0
Dimensions	ndim	A:INT	
Tensor flag	tensor	A:INT	vector, tensor: 1 scalar, matrix: 0
Data Type	type	A:CHAR[ESL]	{ "FLOAT" | "COMPLEX" | "INT" | "CHAR" | "BOOLEAN" | "INTID" }
Length ubid, ubexp	lenb	A:INT	$\text{lenb} = \sum_{\text{ncoo}} \text{mbunit}$
Length udid, udexp	lend	A:INT	$\text{lend} = \sum_{\text{ncoo}} \text{mdunit}$
vmap length	lvmap	A:INT	{ 0 | ncoo x $\text{ndim}^{\text{rank}}$ }
Subscript string length	subslen	A:INT	
No of base units	(i=0; i<ncoo; i++)mbunit[i]	D:INT	
No derived units	(i=0; i<ncoo; i++)mdunit[i]	D:INT	
Base unit ID	(i=0, k=0; i<ncoo; i++) (j=0; j<mbunit[i]; j++)ubid[k++]	D:INT	
Base unit exponents	(i=0, k=0; i<ncoo; i++) (j=0; j<mbunit[i]; j++)ubexp[k++]	D:FLOAT	
Derived unit ID	(i=0,k=0;i<ncoo;i++) (j=0; j<mdunit[i]; j++)udid[k++]	D:INT	
Derived unit exponents	(i=0, k=0; i<ncoo; i++) (j=0; j<mdunit[i]; j++)udexp[k++]	D:FLOAT	
Subscripts	(i=0; i<subslen; i++)subscr[i]	D:CHAR[1]	"xxyyxy", "123"
Option: Length subscript[coordinate]	if(subslen > 0) (i=0; i<ncoo; i++)subscnt[i]	D:INT	= No of characters
Option: Offset subscript[coordinate]	if(subslen > 0) (i=0; i<ncoo; i++)subsoffs[i]	D:INT	= [0; subslen-1]

Option: Vmaps	if(lvmap > 0)
	(i=0; i<ncoo; i++){
	(j=0; j<ndim; j++){
	(k=0; k<ndim; k++){
	…
	(n=0; n<ndim; n++){
	vmap[i][j][k][…][n]
	}
	…
	}
	}
	}
		D:FLOAT	ncoo = no of vmaps
			1st axis
			2nd axis
			…
			rankth axis
			size: ncoo x $\text{ndim}^{\text{rank}}$

Notes to the subscripts (*subscr*):

The subscripts are stored on the character array *subscr*. A subscript can be assigned to any coordinate and could be used as a suffix of the variable name (see example below).

varnam	parameter	subscripts	displayed
Stress	*nrank = 2* *ndim = 2* *ncoo = 3*	*xx, xy, yy*	Stress xx | Stress xy Stress xy | Stress yy

Notes to the variable classes (*varclass*):

The *varclass* strings can be used to classify the variables for certain purposes, e.g. to assign user-defined precision values for the application of lossy compression filters.

Notes to *varnam* and *varkey*:

The string stored in *varnam* might be used to label the variable (e.g. on screen). These names are not supposed to be unique over all variable groups, whereas the string stored in *varkey* is the identifier of the variable and must be unique over the whole file (see variablegroup block key 32).

Notes to the variable map arrays (*vmap*):

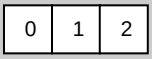
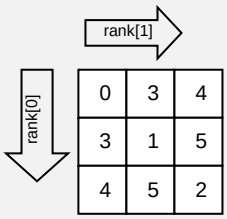
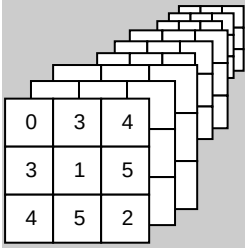
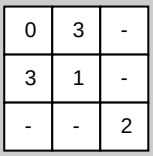
The purpose of the *vmap* arrays is to reconstruct a tensor- or a matrix-variable with its symmetries from the list of coordinates stored in the ERF result block. The result variable $\mathbf{R}$ is the sum over the product of the coordinates c_i and the *vmap* matrices $\mathbf{M_i}$:

$$\mathbf{R} = \sum_{\mathbf{i=1}}^{\mathbf{n}} c_i \mathbf{M_i}$$

The following example shows how vmap arrays can be used to define a symmetric plain strain tensor:

$$\begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{xy} & \varepsilon_{yy} \end{bmatrix} = \varepsilon_{xx} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \varepsilon_{yy} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} + \varepsilon_{xy} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Examples of variable definitions:

variable	category	definition	rank	ncoo	ndim	non-zero	vmap arrays	tensor	illustration
mass	scalar	m	0	1	n/a	1	n/a	0	
velocity	vector	$\vec{v} = v_i \vec{e}_i$	1	3	3	3	$c_0\begin{bmatrix}1\\0\\0\end{bmatrix} + c_1\begin{bmatrix}0\\1\\0\end{bmatrix} + c_2\begin{bmatrix}0\\0\\1\end{bmatrix}$	1	
stress	symm. tensor	$\vec{\vec{\sigma}} = \sigma_{ij} \vec{e}_i \vec{e}_j$	2	6	3	3^2	$c_0\begin{bmatrix}1&0&0\\0&0&0\\0&0&0\end{bmatrix} + \ldots + c_5\begin{bmatrix}0&0&0\\0&0&1\\0&1&0\end{bmatrix}$	1	
stiffness tensor	symm. tensor	$\vec{C}^4 = C_{ijkl} \vec{e}_i \vec{e}_j \vec{e}_k \vec{e}_l$ with $C_{ijkl} = C_{jikl} = C_{ijlk} = C_{likj}$	4	21	3	3^4	$\sum_{21} c_i \mathbf{M}_i^{(4)}$	1	
damage	array of scalars	$\underline{D} = (d_1, d_2, d_3)^T$	1	3	3	3	$c_0\begin{bmatrix}1\\0\\0\end{bmatrix} + c_1\begin{bmatrix}0\\1\\0\end{bmatrix} + c_2\begin{bmatrix}0\\0\\1\end{bmatrix}$	0	
stiffness matrix	array of scalars	$\underline{C} = \begin{bmatrix}c_0 & c_3 & 0\\ c_3 & c_1 & 0\\ 0 & 0 & c_2\end{bmatrix}$	2	4	3	5	$c_0\begin{bmatrix}1&0&0\\0&0&0\\0&0&0\end{bmatrix} + \ldots + c_3\begin{bmatrix}0&1&0\\1&0&0\\0&0&0\end{bmatrix}$	0	

3.6 Block 32 – Variable Group Definition Block

Name	Variable [= value]	type	comments
Block key	block = 32	A:INT	
Block type	blocktype="variablegroup"	A:CHAR[ESL]	
Group identifier	etypvar	A:CHAR[ESL]	e.g. "SPH-Variables"
No of Variables	nvar	A:INT	
Independent coordinates	ncoo	A:INT	
No of coordinates	(i=0; i<nvar; i++)cnt[i]	D:INT	
Coordinate offset	(i=0; i<nvar; i++)offs[i]	D:INT	
Variable key	(i=0; i<nvar; i++)varkey[i]	D:CHAR[ESL]	"VPS:LagrangianStrain", …

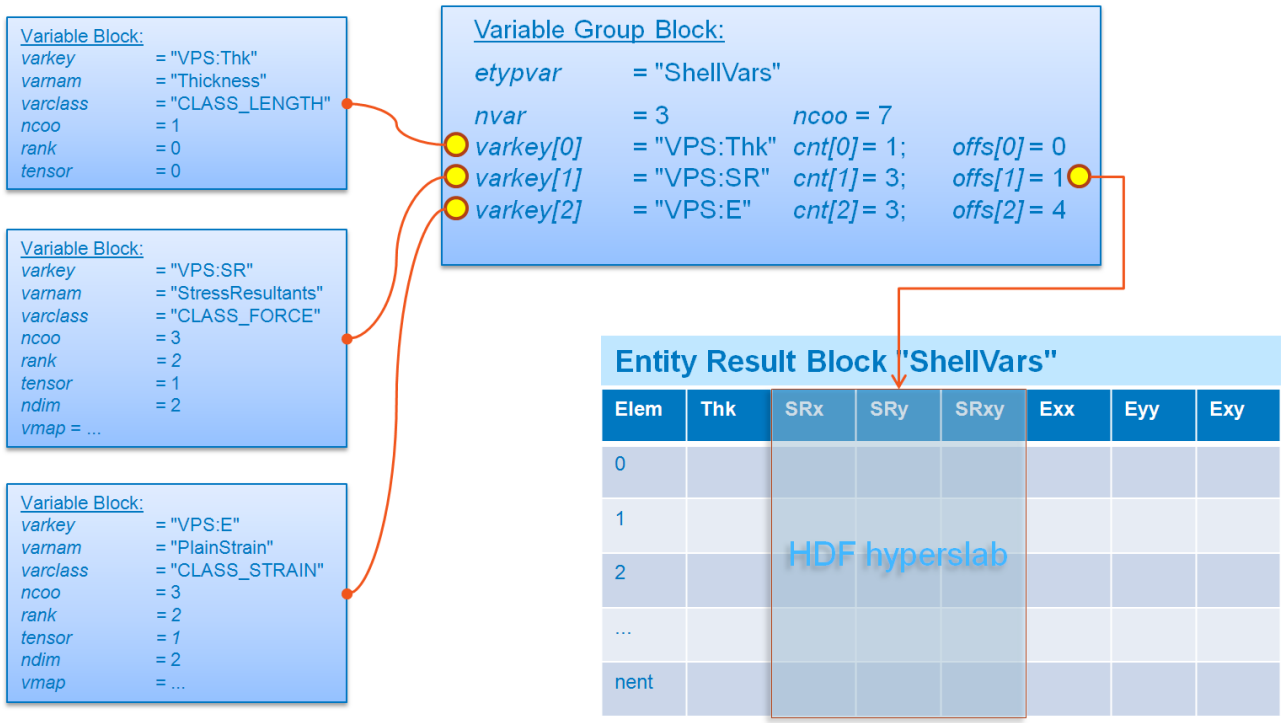
Notes to variable group definitions:

Variable groups can be referenced in the following data blocks:

- result blocks (single- or multistate *entity- or element results*),
- index definition blocks (*indices*),
- connectivities blocks (*fparam* values).

In those blocks the coordinates of a requested variable can be addressed by using the arrays *cnt* and *offs*, whereas *cnt* contains the number of coordinates and *offs* the offset of the first coordinate of the variable. In the data blocks all coordinates of a variable are consistently stored without gap.

The string *varkey* is the identifier of the variable, which is defined in a variable definition block (block key 30). The following example shows how the results for a certain variable can be found in the entity result block:



3.7 Block 40 – Text Block

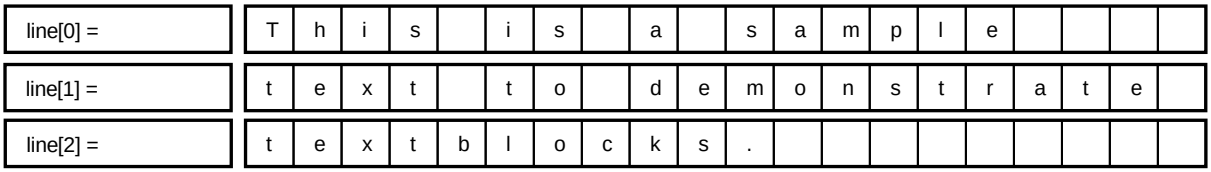
Name	Variable [= value]	type	comments
Block key	block = 40	A:INT	
Generic block type	blocktype = "text"	A:CHAR[ESL]	
Textblock type	etyptext	A:CHAR[ESL]	user-defined, e.g. INPUT
Number of lines	nlines	A:INT	nlines > 0
Line length	length	A:INT	length > 0
Single-State Index	…	…	
Text	(i=0; i<nlines; i++)line[i]	D:CHAR[length]	

Sample:

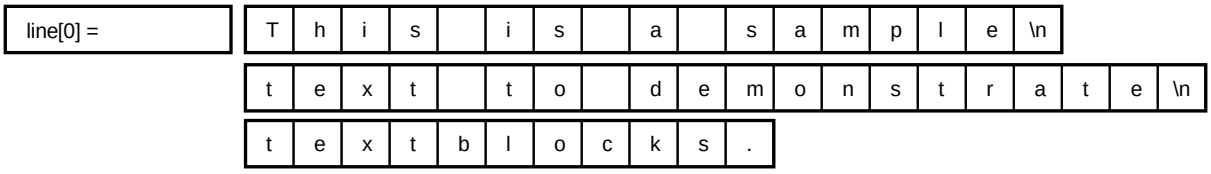
The following text

```
This is a sample
text to demonstrate
textblocks.
```

could either be stored as multiple lines – each with the same length:



Or, alternatively, the text could be more compactly stored in a single line and using some c-style escape sequences for formatting, e.g. end-of-line characters:



3.8 Block 100 – Parts Block

Name	Variable [= value]	type	comments
Block key	block = 100	A:INT	
Generic block type	blocktype = "parts"	A:CHAR[ESL]	
Part entity type	etyppart	A:CHAR[ESL]	
Number of parts	npart	A:INT	
Length of title strings	length	A:INT	length ≥ 0
Option switch to suppress optional arrays	off	A:INT	Bit | Value | Array 0 | 1 | ~~mid~~ 1 | 2 | ~~mtyp~~ 2 | 4 | ~~pcol~~
Single-State Index	…	…	
Part ID	(i=0; i<npart; i++)pid[i]	D:INTID	
Part title	(i=0; i<npart; i++)title[i]	D:CHAR[length]	
Option: Material ID	if(!(off & 1)) (i=0; i<npart; i++) mid[i]	D:INTID	
Option: Material type	if(!(off & 2)) (i=0; i<npart; i++) mtyp[i]	D:INT	
Option: Color Index	if(!(off & 4)) (i=0; i<npart; i++) pcol[i]	D:INT	

3.9 Block 2xx – User Identifiers Blocks

3.9.1 Block 200 – Attributes Block

Name	Variable [= value]	type	comments
Block key	block = 200	A:INT	
Generic block type	blocktype = "attributes"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	e.g. PART, SHELL, NODE, …
Number of entities	nent	A:INT	
Length of title strings	length	A:INT	length ≥ 0
Number of user ids	nuid	A:INT	nuid ≥ 0
Single-State Index	**…**	**…**	
Entity ID	(i=0; i<nent; i++)entid[i]	D:INTID	
Entity title	(i=0; i<nent; i++)title[i]	D:CHAR[length]	
User ID j of ent. i	(i=0; i<nent; i++) (j=0; j<nuid; j++) uid[i][j]	D:INTID	

3.9.2 Block 250 – Identifiers Block

Name	Variable [= value]	type	comments
Block key	block = 250	A:INT	
Generic block type	blocktype = "identifiers"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	e.g. PART, SHELL, NODE, …
Number of entities	nent	A:INT	
Length of title strings	length	A:INT	length ≥ 0
Number of user ids	nuid	A:INT	nuid ≥ 0
Single-State Index	**…**	**…**	
Entity ID	(i=0; i<nent; i++)entid[i]	D:INTID	
Entity title	(i=0; i<nent; i++)title[i]	D:CHAR[length]	
User ID j of ent. i	(i=0; i<nent; i++) (j=0; j<nuid; j++) uid[i][j]	D:INTID	

Note:

The "identifiers" block is a clone of the "attributes" block with a different key and block type. The purpose of this block is to store user identifiers for modular solver I/O support.

Modular I/O is a solver feature to enable the user to easily exchange or include model components without taking care of the uniqueness of the user entity IDs (nodes, elements, parts etc.). The solver internally uses sequential numbers to avoid clashing ID's between the main and the sub-models. These sequential numbers are written as entity ID's in the ERF result blocks and can be retranslated into user ID's by using the information stored in the "identifiers" blocks.

3.10 Block 300 – Element Connectivity Block

name	Variable [= value]	type	comments
Block key	block = 300	A:INT	
Generic block type	blocktype="connectivities"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	user-defined, e.g. "ELEMENT"
Element type	etypelem	A:CHAR[ESL]	user-defined, e.g. "SHELL"
Part entity type	etyppart	A:CHAR[ESL]	user-defined, e.g. "PART"
Node entity type	etypnode	A:CHAR[ESL]	user-defined, e.g. "NODE"
No of elements	nele	A:INT	
No of nodes per element	npele	A:INT	
Element dimensions	ndim	A:INT	point=0, bar=1, shell=2,solid=3
No of integer parameter	nbint	A:INT	nbint ≥ 0
No of float parameters	nbfloat	A:INT	nbfloat ≥ 0
Option: Variable to declare fparam values	if(nbfloat > 0) etypvar	A:CHAR[ESL]	ncoo(etypvar) = nbfloat
Single-State Index	**…**	**…**	
Element ID	(i=0; i<nele; i++)idele[i]	D:INTID	
Part ID	(i=0; i<nele; i++)pid[i]	D:INTID	
Node connectivity	(i=0; i<nele; i++) (j=0; j<npele; j++) ic[i][j]	D:INTID	
Integer parameter	(i=0; i<nele; i++) (j=0; j<nbint; j++) iparam[i][j]	D:INT	
Floating point parameter	(i=0; i<nele; i++) (j=0; j<nbfloat; j++) fparam[i][j]	D:FLOAT	

3.11 Block 400 – Collector Block

name	Variable [= value]	type	Comments
Block key	block = 400	A:INT	
Generic block type	blocktype = "collector"	A:CHAR[ESL]	
Collector entity type	etypcoll	A:CHAR[ESL]	
No of Collectors	ncoll	A:INT	
No of entity types	ntyp	A:INT	
Total no of entities	nenttot	A:INT	$nenttot = \sum_{ncoll} \sum_{ntyp} nent$
Single-State Index	...	...	
Collector ID	(i=0; i<ncoll; i++)idcoll[i]	D:INTID	
Entity type	(i=0; i<ntyp; i++)etyp[i]	D:CHAR[ESL]	e.g. NODE, SHELL, …
No of entities per collector	(i=0; i<ncoll; i++) (j=0; j<ntyp; j++) nent[i][j]	D:INT	
Entity ID	(i=0, n=0; i<ncoll; i++) (j=0; j<ntyp; j++) (k=0; k<nent[i][j]; k++) entid[n++]	D:INTID	

Note: A collector can contain other collectors, i.e., the definition can be recursive.

3.12 Block 500 – Properties Block

Name	Variable [= value]	type	comments
Block key	block = 500	A:INT	
Generic block type	blocktype = "properties"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	e.g. PART, SHELL, NODE, …
Property type	etypprop	A:CHAR[ESL]	
Zone type	etypzone	A:CHAR[ESL]	e.g. NONE, PLY, LAYER
Zone ID	zoneid	A:INTID	→ unique ID of etypzone
No. of entities	nent	A:INT	
No. of identifiers	nbid	A:INT	nbid ≥ 0
No. of int parameters	nbint	A:INT	nbint ≥ 0
No. of float parameters	nbfloat	A:INT	nbfloat ≥ 0
No. of strings	nbstring	A:INT	
Length of strings	length	A:INT	length ≥ 0
Single-State Index	*…*	*…*	
Entity ID	(i=0; i<nent; i++)entid[i]	D:INTID	
Identifiers	(i=0; i<nent; i++) (j=0; j<nbid; j++) idparam[i][j]	D:INTID	
Integer parameter	(i=0; i<nent; i++) (j=0; j<nbint; j++) iparam[i][j]	D:INT	
Floating point parameter	(i=0; i<nent; i++) (j=0; j<nbfloat; j++) fparam[i][j]	D:FLOAT	
String parameter	(i=0; i<nent; i++) (j=0; j<nbstring; j++) cparam[i][j]	D:CHAR[length]	

3.13 Block 10xx – Entity Result Block

3.13.1 Block 1000 – Entity Result Block for Real Numbers

name	Variable [= value]	type	comments
Block key	block = 1000	A:INT	
Generic block type	blocktype="entityresults"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
Variable group	etypvar	A:CHAR[ESL]	→ variable group block
Frame type	etypframe	A:CHAR[ESL]	e.g. FRAME, FRAME2D
Zone type	etypzone	A:CHAR[ESL]	e.g. NONE, PLY, LAYER
No of entities	nent	A:INT	
No of coordinates	ncoo	A:INT	→ variable block
Zone ID	zoneid	A:INTID	→ unique ID of etypzone
DMP reduction switch	dmpswitch = { 0 | 1 }	A:INT	0: off, 1: on
Frame control switch	fswitch = { 0 | 1 }	A:INT	= 0 → use fidglob = 1 → use fid
Single-State Index	**…**	**…**	
Entity ID	(i=0; i<nent; i++)entid[i]	D:INTID	

Option: frame IDs	if(fswitch == 0)fidglob
	
	if(fswitch == 1)
	(i=0; i<nent; i++)fid[i]
		D:INTID	= 0 → global cartesian
			> 0 → frame id
			< 0 → local element frame

Option: DMP reduction operators	if(dmpswitch == 1) (i=0; i<ncoo; i++) operator[i]	D:CHAR[ESL]	e.g. SUM, OR, MAX
Option: DMP domain weights	if(dmpswitch == 1) (i=0; i<nent; i++) weight[i]	D:FLOAT	
Result coordinates	(i=0; i<nent; i++) (j=0; j<ncoo; j++) res[i][j]	D:FLOAT	

3.13.2 Block 1001 – Entity Result Block for Complex Numbers

name	Variable [= value]	type	comments
Block key	block = 1001	A:INT	
Generic block type	blocktype="entityresults"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
Variable group	etypvar	A:CHAR[ESL]	→ variable group block
Frame type	etypframe	A:CHAR[ESL]	e.g. FRAME, FRAME2D
Zone type	etypzone	A:CHAR[ESL]	e.g. NONE, PLY, LAYER
No of entities	nent	A:INT	
No of coordinates	ncoo	A:INT	→ variable block
Zone ID	zoneid	A:INTID	→ unique ID of etypzone
DMP reduction switch	dmpswitch = { 0 | 1 }	A:INT	0: off, 1: on
Frame control switch	fswitch = { 0 | 1 }	A:INT	= 0 → use fidglob = 1 → use fid
Single-State Index	*…*	*…*	
Entity ID	(i=0; i<nent; i++)entid[i]	D:INTID	

Option: frame IDs	if(fswitch == 0)fidglob
	
	if(fswitch == 1)
	(i=0; i<nent; i++)fid[i]
		D:INTID	= 0 → global cartesian
			> 0 → frame id
			< 0 → local element frame

Option: DMP reduction operators	if(dmpswitch == 1) (i=0; i<ncoo; i++) operator[i]	D:CHAR[ESL]	e.g. SUM, OR, MAX
Option: DMP domain weights	if(dmpswitch == 1) (i=0; i<nent; i++) weight[i]	D:FLOAT	
Result coordinates	(i=0; i<nent; i++) (j=0; j<ncoo; j++) (k=0; k<2; k++) res[i][j][k]	D:FLOAT	k=0: real part k=1: imaginary part

3.13.3 Block 1050 – Multi-State Entity Result Block for Real Numbers

name	Variable [= value]	type	comments
Block key	block = 1050	A:INT	
Generic block type	blocktype="multientityresults"	A:CHAR[ESL]	
Series name	series	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
Variable group	etypvar	A:CHAR[ESL]	→ variable group block
Frame type	etypframe	A:CHAR[ESL]	e.g. FRAME, FRAME2D
Zone type	etypzone	A:CHAR[ESL]	e.g. NONE, PLY, LAYER
No of entities	nent	A:INT	
No of coordinates	ncoo	A:INT	→ variable block
Zone ID	zoneid	A:INTID	→ unique ID of etypzone
DMP reduction switch	dmpswitch = { 0 | 1 }	A:INT	0: off, 1: on
Frame control switch	fswitch = { 0 | 1 }	A:INT	= 0 → use fidglob = 1 → use fid
Multi-State Index	**…**	**…**	
Entity ID	(i=0; i<nent; i++)entid[i]	D:INTID	

Option: frame IDs	if(fswitch == 0)fidglob
	
	if(fswitch == 1)
	(i=0; i<nent; i++)fid[i]
		D:INTID	= 0 → global cartesian
			> 0 → frame id
			< 0 → local element frame

Option: DMP reduction operators	if(dmpswitch == 1) (i=0; i<ncoo; i++) operator[i]	D:CHAR[ESL]	e.g. SUM, OR, MAX
Option: DMP domain weights	if(dmpswitch == 1) (i=0; i<nent; i++) weight[i]	D:FLOAT	
Result coordinates	(i=0; i<nstate; i++) (j=0; j<nent; j++) (k=0; k<ncoo; k++) res[i][j][k]	D:FLOAT	

3.13.4 Block 1051 – Multi-State Entity Result Block for Complex Numbers

name	Variable [= value]	type	comments
Block key	block = 1050	A:INT	
Generic block type	blocktype="multientityresults"	A:CHAR[ESL]	
Series name	series	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
Variable group	etypvar	A:CHAR[ESL]	→ variable group block
Frame type	etypframe	A:CHAR[ESL]	e.g. FRAME, FRAME2D
Zone type	etypzone	A:CHAR[ESL]	e.g. NONE, PLY, LAYER
No of entities	nent	A:INT	
No of coordinates	ncoo	A:INT	→ variable block
Zone ID	zoneid	A:INTID	→ unique ID of etypzone
DMP reduction switch	dmpswitch = { 0 | 1 }	A:INT	0: off, 1: on
Frame control switch	fswitch = { 0 | 1 }	A:INT	= 0 → use fidglob = 1 → use fid
Multi-State Index	**…**	**…**	
Entity ID	(i=0; i<nent; i++)entid[i]	D:INTID	

Option: frame IDs	if(fswitch == 0)fidglob
	
	if(fswitch == 1)
	(i=0; i<nent; i++)fid[i]
		D:INTID	= 0 → global cartesian
			> 0 → frame id
			< 0 → local element frame

Option: DMP reduction operators	if(dmpswitch == 1) (i=0; i<ncoo; i++) operator[i]	D:CHAR[ESL]	e.g. SUM, OR, MAX
Option: DMP domain weights	if(dmpswitch == 1) (i=0; i<nent; i++) weight[i]	D:FLOAT	
Result coordinates	(i=0; i<nstate; i++) (j=0; j<nent; j++) (k=0; k<ncoo; k++) (l=0; l<2; l++) res[i][j][k][l]	D:FLOAT	l=0: real part l=1: imaginary part

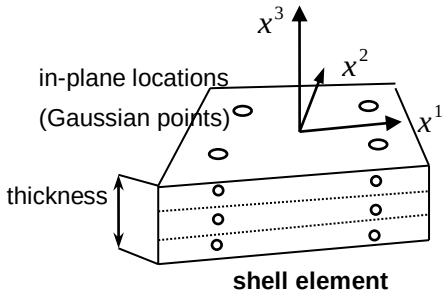
3.14 Block 11xx – Intra-Elemental Result Block

3.14.1 Introduction

The intra-elemental result block contains local element results at elemental intra-location points. An intra-location point can be any location of the elemental domain (even outside), e.g. Gaussian integration points, nodal positions or any other user-defined position. The intra-locations are defined by parametric coordinates. The output results are supposed to be functions of these parametric coordinates (iso-parametric concept).

Example: shell element, output of a symmetric plane stress tensor

- result: 4 components, 3 coordinates (variable definition: rank = 2, ndim = 2, ncoo = 3)
- elemental intra-locations: 4 in-plane gauss points over 3 out-of-plane layer points (numl=12, ndim=3)
- output in local element frame (fid = -1)



$$\vec{\vec{\sigma}} = \sigma_{\alpha\beta}\vec{a}_\alpha\vec{a}_\beta \ , \quad \alpha, \beta = 1, 2$$

etypvar tensor coordinates; local element frame

out-of-plane locations
(integration points through thickness)

(k=0; k<numl; k++)	x^1	x^2	x^3
0	-α	-α	-1
1	+α	-α	-1
2	+α	+α	-1
3	-α	+α	-1
4	-α	-α	0
5	+α	-α	0
6	+α	+α	0
7	-α	+α	0
8	-α	-α	+1
9	+α	-α	+1
10	+α	+α	+1
11	-α	+α	+1

3.14.2 Block 1100 – Intra-Elemental Result Block for Real Numbers

name	Variable [= value]	type	comments
Block key	block = 1100	A:INT	
Generic block type	blocktype="elementresults"	A:CHAR[ESL]	
Element type	etypelem	A:CHAR[ESL]	
Variable group	etypvar	A:CHAR[ESL]	→ variable group block
Frame type	etypframe	A:CHAR[ESL]	e.g. FRAME, FRAME2D
Zone type	etypzone	A:CHAR[ESL]	e.g. NONE, PLY, LAYER
No of elements	nele	A:INT	
No of coordinates	ncoo	A:INT	→ variable block
No of intra-locations	numl	A:INT	
Intra-location dimensions	ndim	A:INT	
Zone ID	zoneid	A:INTID	→ unique ID of etypzone
DMP reduction switch	dmpswitch = { 0 | 1 }	A:INT	0: off, 1: on
Frame control switch	fswitch = { 0 | 1 | 2 }	A:INT	= 0 → use fidglob = 1 → single fid per element = 2 → fid per intra-location
Single-State Index	*…*	*…*	
Entity ID	(i=0; i<nele; i++)idele[i]	D:INTID	

Option: frame IDs	if(fswitch == 0)fidglob
	
	if(fswitch == 1)
	(i=0; i<nele; i++)fid[i]
	
	if(fswitch == 2)
	(i=0; i<nele; i++)
	(j=0; j<numl; j++)
	fid[i][j]
		D:INTID	= 0 → global cartesian
			> 0 → frame id
			< 0 → local element frame

Option: DMP reduction operators	if(dmpswitch == 1) (i=0; i<ncoo; i++) operator[i]	D:CHAR[ESL]	e.g. SUM, OR, MAX
Option: DMP domain weights	if(dmpswitch == 1) (i=0; i<nele; i++) weight[i]	D:FLOAT	
Isoparametric coordinates of intra-locations	(i=0; i<numl; i++) (j=0; j<ndim; j++) eloc[i][j]	D:FLOAT	
Result coordinates	(i=0; i<nele; i++) (j=0; j<numl; j++) (k=0; k<ncoo; k++) res[i][j][k]	D:FLOAT	

3.14.3 Block 1101 – Intra-Elemental Result Block for Complex Numbers

name	Variable [= value]	type	comments
Block key	block = 1101	A:INT	
Generic block type	blocktype="elementresults"	A:CHAR[ESL]	
Element type	etypelem	A:CHAR[ESL]	
Variable group	etypvar	A:CHAR[ESL]	→ variable group block
Frame type	etypframe	A:CHAR[ESL]	e.g. FRAME, FRAME2D
Zone type	etypzone	A:CHAR[ESL]	e.g. NONE, PLY, LAYER
No of elements	nele	A:INT	
No of coordinates	ncoo	A:INT	→ variable block
No of intra-locations	numl	A:INT	
Intra-location dimensions	ndim	A:INT	
Zone ID	zoneid	A:INTID	→ unique ID of etypzone
DMP reduction switch	dmpswitch = { 0 | 1 }	A:INT	0: off, 1: on
Frame control switch	fswitch = { 0 | 1 | 2 }	A:INT	= 0 → use fidglob = 1 → single fid per element = 2 → fid per intra-location
Single-State Index	**…**	**…**	
Entity ID	(i=0; i<nele; i++)idele[i]	D:INTID	

Option: frame IDs	if(fswitch == 0)fidglob
	
	if(fswitch == 1)
	(i=0; i<nele; i++)fid[i]
	
	if(fswitch == 2)
	(i=0; i<nele; i++)
	(j=0; j<numl; j++)
	fid[i][j]
		D:INTID	= 0 → global cartesian
			> 0 → frame id
			< 0 → local element frame

Option: DMP reduction operators	if(dmpswitch == 1) (i=0; i<ncoo; i++) operator[i]	D:CHAR[ESL]	e.g. SUM, OR, MAX
Option: DMP domain weights	if(dmpswitch == 1) (i=0; i<nele; i++) weight[i]	D:FLOAT	
Isoparametric coordinates of intra-locations	(i=0; i<numl; i++) (j=0; j<ndim; j++) eloc[i][j]	D:FLOAT	
Result coordinates	(i=0; i<nele; i++) (j=0; j<numl; j++) (k=0; k<ncoo; k++) (l=0; l<2; l++) res[i][j][k][l]	D:FLOAT	l=0: real part l=1: imaginary part

3.14.4 Block 1150 – Multi-State Intra-Elemental Result Block for Real Numbers

name	Variable [= value]	type	comments
Block key	block = 1150	A:INT	
Generic block type	blocktype="multielementresults"	A:CHAR[ESL]	
Series name	series	A:CHAR[ESL]	
Element type	etypelem	A:CHAR[ESL]	
Variable group	etypvar	A:CHAR[ESL]	→ variable group block
Frame type	etypframe	A:CHAR[ESL]	e.g. FRAME, FRAME2D
Zone type	etypzone	A:CHAR[ESL]	e.g. NONE, PLY, LAYER
No of elements	nele	A:INT	
No of coordinates	ncoo	A:INT	→ variable block
No of intra-locations	numl	A:INT	
Intra-location dimensions	ndim	A:INT	
Zone ID	zoneid	A:INTID	→ unique ID of etypzone
DMP reduction switch	dmpswitch = { 0 | 1 }	A:INT	0: off, 1: on
Frame control switch	fswitch = { 0 | 1 | 2 }	A:INT	= 0 → use fidglob = 1 → single fid per element = 2 → fid per intra-location
Multi-State Index	**…**	**…**	
Entity ID	(i=0; i<nele; i++)idele[i]	D:INTID	

Option: frame IDs	if(fswitch == 0)fidglob
	
	if(fswitch == 1)
	(i=0; i<nele; i++)fid[i]
	
	if(fswitch == 2)
	(i=0; i<nele; i++)
	(j=0; j<numl; j++)
	fid[i][j]
		D:INTID	= 0 → global cartesian
			> 0 → frame id
			< 0 → local element frame

Option: DMP reduction operators	if(dmpswitch == 1) (i=0; i<ncoo; i++) operator[i]	D:CHAR[ESL]	e.g. SUM, OR, MAX
Option: DMP domain weights	if(dmpswitch == 1) (i=0; i<nele; i++) weight[i]	D:FLOAT	
Isoparametric coordinates of intra-locations	(i=0; i<numl; i++) (j=0; j<ndim; j++) eloc[i][j]	D:FLOAT	
Result coordinates	(i=0; i<nstate; i++) (j=0; j<nele; j++) (k=0; k<numl; k++) (l=0; l<ncoo; l++) res[i][j][k][l]	D:FLOAT	

3.14.5 Block 1151 – Multi-State Intra-Elemental Result Block for Complex Numbers

name	Variable [= value]	type	comments
Block key	block = 1151	A:INT	
Generic block type	blocktype="multielementresults"	A:CHAR[ESL]	
Series name	series	A:CHAR[ESL]	
Element type	etypelem	A:CHAR[ESL]	
Variable group	etypvar	A:CHAR[ESL]	→ variable group block
Frame type	etypframe	A:CHAR[ESL]	e.g. FRAME, FRAME2D
Zone type	etypzone	A:CHAR[ESL]	e.g. NONE, PLY, LAYER
No of elements	nele	A:INT	
No of coordinates	ncoo	A:INT	→ variable block
No of intra-locations	numl	A:INT	
Intra-location dimensions	ndim	A:INT	
Zone ID	zoneid	A:INTID	→ unique ID of etypzone
DMP reduction switch	dmpswitch = { 0 | 1 }	A:INT	0: off, 1: on
Frame control switch	fswitch = { 0 | 1 | 2 }	A:INT	= 0 → use fidglob = 1 → single fid per element = 2 → fid per intra-location
Multi-State Index	**…**	**…**	
Entity ID	(i=0; i<nele; i++)idele[i]	D:INTID	

Option: frame IDs	if(fswitch == 0)fidglob
	
	if(fswitch == 1)
	(i=0; i<nele; i++)fid[i]
	
	if(fswitch == 2)
	(i=0; i<nele; i++)
	(j=0; j<numl; j++)
	fid[i][j]
		D:INTID	= 0 → global cartesian
			> 0 → frame id
			< 0 → local element frame

Option: DMP reduction operators	if(dmpswitch == 1) (i=0; i<ncoo; i++) operator[i]	D:CHAR[ESL]	e.g. SUM, OR, MAX
Option: DMP domain weights	if(dmpswitch == 1) (i=0; i<nele; i++) weight[i]	D:FLOAT	
Isoparametric coordinates of intra-locations	(i=0; i<numl; i++) (j=0; j<ndim; j++) eloc[i][j]	D:FLOAT	
Result coordinates	(i=0; i<nstate; i++) (j=0; j<nele; j++) (k=0; k<numl; k++) (l=0; l<ncoo; l++) (m=0; m<2; m++) res[i][j][k][l][m]	D:FLOAT	m=0: real part m=1: imaginary part

3.15 Block 21xx – Activation Flag Block

3.15.1 Block 2100 - Activation Flag Block

name	Variable [= value]	type	comments
Block key	block = 2100	A:INT	
Generic block type	blocktype = "activflags"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
No of flagged entities	nent	A:INT	
Single-State Index	**…**	**…**	
Entity ID	(i=0; i<nent; i++)entid[i]	D:INTID	
Status flag	(i=0; i<nent; i++)status[i]	D:INT	

3.15.2 Block 2150 – Multi-State Activation Flag Block

name	Variable [= value]	type	comments
Block key	block = 2150	A:INT	
Generic block type	blocktype="multiactivflags"	A:CHAR[ESL]	
Series name	series	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
No of flagged entities	nent	A:INT	
Multi-State Index	**…**	**…**	
Entity ID	(i=0; i<nent; i++)entid[i]	D:INTID	
Status flag	(i=0; i<nstate; i++) (j=0; j<nent; j++) status[i][j]	D:INT	

3.16 Block 10xxx – Matrix Block

3.16.1 Terms and Definitions

Matrix:

$$\underline{A} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$$

Triangular and Symmetric Matrices:

$val(k) = a_{ij}$, $k = 1, \ldots, n(n+1)/2$ (with n = nsize)

matrix	matrixtype	definition	Example
lower triangular matrix	1	$i = 1, \ldots, n$ $j = 1, \ldots, i$ $k = i(i-1)/2 + j$	1 - - 2 3 - 4 5 6
upper triangular matrix	2	$i = 1, \ldots, n$ $j = i, \ldots, n$ $k = (i-1)(2n-i)/2 + j$	1 2 3 - 4 5 - - 6
lower triangular matrix transposed	3	$j = 1, \ldots, n$ $i = 1, \ldots, j$ $k = j(j-1)/2 + i$	1 2 4 - 3 5 - - 6
upper triangular matrix transposed	4	$j = 1, \ldots, n$ $i = j, \ldots, n$ $k = (j-1)(2n-j)/2 + i$	1 - - 2 4 - 3 5 6
symmetric matrix (lower)	5	$i = 1, \ldots, n$ $j = 1, \ldots, i$ $k = i(i-1)/2 + j$	1 2 4 2 3 5 4 5 6
symmetric matrix (upper)	6	$i = 1, \ldots, n$ $j = i, \ldots, n$ $k = (i-1)(2n-i)/2 + j$	1 2 3 2 4 5 3 5 6

Sparse matrix CRS (Compressed Sparse Row) Storage Scheme:

$val(k) = a_{ij}$, $i, j = 1 \ldots nrow$, $k = 1 \ldots nnz$

$colind(k) = j$, $rowptr(i) \le k < rowptr(i+1)$, $rowptr(nrow+1) = nnz + 1$

3.16.2 Block 10000 – Dense Matrix Block for Real Numbers

name	Variable [= value]	type	comments
Block key	block = 10000	A:INT	
Generic block type	blocktype="densematrix"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
Entity ID	entid	A:INTID	
Variable group	etypvar	A:CHAR[ESL]	
No of rows	nrow	A:INT	
No of columns	ncolumn	A:INT	
Single-State Index	*...*	*...*	
Matrix values	(i=0; i<nrow; i++) (j=0; j<ncolumn; j++) val[i][j]	D:FLOAT	

3.16.3 Block 10001 – Dense Matrix Block for Complex Numbers

name	Variable [= value]	type	comments
Block key	block = 10001	A:INT	
Generic block type	blocktype="densematrix"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
Entity ID	entid	A:INTID	
Variable group	etypvar	A:CHAR[ESL]	
No of rows	nrow	A:INT	
No of columns	ncolumn	A:INT	
Single-State Index	*...*	*...*	
Matrix values	(i=0; i<nrow; i++) (j=0; j<ncolumn; j++) (k=0; k<2; k++) val[i][j][k]	D:FLOAT	k=0: real part k=1: imaginary part

3.16.4 Block 10100 – Triangular Matrix Block for Real Numbers

name	Variable [= value]	type	comments
Block key	block = 10100	A:INT	
Generic block type	blocktype="triangularmatrix"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
Entity ID	entid	A:INTID	
Variable group	etypvar	A:CHAR[ESL]	
No of rows or columns	nsize	A:INT	
Matrix type	matrixtype	A:INT	see table above
Single-State Index	...	...	
Matrix values	(i=0; i<nsize*(nsize+1)/2; i++) val[i]	D:FLOAT	

3.16.5 Block 10101 – Triangular Matrix Block for Complex Numbers

name	Variable [= value]	type	comments
Block key	block = 10101	A:INT	
Generic block type	blocktype="triangularmatrix"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
Entity ID	entid	A:INTID	
Variable group	etypvar	A:CHAR[ESL]	
No of rows or columns	nsize	A:INT	
Matrix type	matrixtype	A:INT	see table above
Single-State Index	...	...	

Matrix values	(i=0; i<nsize*(nsize+1)/2; i++)
	  (j=0; j<2; j++)
	  val[i][j]
		D:FLOAT	j=0: real part
			j=1: imaginary part

3.16.6 Block 10200 – Sparse CRS Matrix Block for Real Numbers

name	Variable [= value]	type	comments
Block key	block = 10200	A:INT	
Generic block type	blocktype="sparsematrix"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
Entity ID	entid	A:INTID	
Variable group	etypvar	A:CHAR[ESL]	
No of rows = columns	nrow	A:INT	
No of non-zero values	nnz	A:LONG	
Single-State Index	...	...	
Column index	(i=0; i<nnz; i++)colind[i]	D:INT	
Row pointer	(i=0; i<nrow+1; i++)rowptr[i]	D:LONG	
Non-zero matrix values	(i=0; i<nnz; i++)val[i]	D:FLOAT	

3.16.7 Block 10201 – Sparse CRS Matrix Block for Complex Numbers

name	Variable [= value]	type	comments
Block key	block = 10201	A:INT	
Generic block type	blocktype="sparsematrix"	A:CHAR[ESL]	
Entity type	etyp	A:CHAR[ESL]	
Entity ID	entid	A:INTID	
Variable group	etypvar	A:CHAR[ESL]	
No of rows = columns	nrow	A:INT	
No of non-zero values	nnz	A:LONG	
Single-State Index	...	...	
Column index	(i=0; i<nnz; i++)colind[i]	D:INT	
Row pointer	(i=0; i<nrow+1; i++)rowptr[i]	D:LONG	
Non-zero matrix values	(i=0; i<nnz; i++) (j=0; j<2; j++) val[i][j]	D:FLOAT	j=0: real part j=1: imaginary part