| ESI Group | Functional specification | N°: 4 |
|---|---|---|

# ERF-HDF5 Specification
## Version 1.2

**Business or Project N°: CSM 2011: ERF - ESI Result Format**

| Rev. | WRITTEN BY | | | CHECKED BY | | | APPROVED BY | | |
|---|---|---|---|---|---|---|---|---|---|
| | Name(s) | Date | Visa | Name (s) | Date | Visa | Name (s) | Date | Visa |
| **A** | AFL | Jan'11 | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

**Filing:**

Name of the filing responsible: **AFL**
Date : Jan'11
Visa :
This document is organized by:
- Pages: 35
- Attachments: -

**Consultation:**

Free : x
Available at the internal website : x
ESI Group only :
Diffusion list only :
Others ESI Group's Subsidiaries:

| Reviews | Reviewed pages |
|---|---|
| **A** | Original document |
| | |
| | |

**DISTRIBUTION :**

| Company | Name(s) | Total | Partial |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

**ESI Group S.A.**
**Paris –** Adresse du Siège Social : 6 Rue Hamelin – BP 2008-16 – 75761 PARIS CEDEX 16 - Tél. : 33 (0)1.53.65.14.14 – Fax : 33 (0)1.53.65.14.12
**Rungis** PARC D AFFAIRES SILIC - 99 RUE DES SOLETS – BP 80112 – 94513 RUNGIS CEDEX - Tel : 33 (0)1.41.73.58.00 – Fax : 33 (0)1.46.87.72.02
**Lyon :** "Le Discover" – 84 Bd. Vivier Merle – 69485 LYON CEDEX 03 – Tél. : 33 (0)4.78.14.12.00 – Fax : 33 (0)4.78.14.12.02
**Aix en Provence :** 5 Parc Club du Golf - 13856 AIX-EN-PROVENCE CEDEX 3 – Tél. : 33 (0)4.42.97.65.30 – Fax : 33 (0)4.42.97.65.39
**Compiègne :** 20 Rue du Fonds Pernant – Immeuble Thalassa – 60471 COMPIEGNE CEDEX - Tél. : 33 (0)3.44.30.43.60 – Fax : 33 (0)1.44.86.87.77
**Montpellier :** Parc club du Millénaire – Bat 15 - 1025 Rue Henri Becquerel – 34000 MONTPELLIER - Tél. : +33 (0)4 67 64 50 43
RCS PARIS B381 080 225 000 26

## *Document status: REVIEWED*


| | |
|---|---|
| Prepared by : | Dr. Andreas Floss |
| Phone number : | +41 21 6938321 |
| E-mail address : | afl@esi-group.com |

Reviewed by :

| <reviewer> | <email> | <phone number> |
|---|---|---|
| Dr. Raymond Ni | rni@esi-group.com | |
| Thorsten Queckbörner | tqu@esigmbh.de | |

CSM Solver Development

# ERF-HDF5 Specification
# Version 1.2

ESI Software

January 2011
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 blocks are divided into three categories:
- constant data blocks,
- single-state variable blocks,
- multi-state variable blocks.

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

Multi-state blocks contain extendable datasets (see HDF5 documentation) to store multiple states of the same data (results, coordinates, etc.) in a single dataset. These extendable datasets have – unlike datasets with fixed dimensions – one unlimited dimension to store an arbitrary number of simulation states. The main purpose of multi-state blocks is to write, handle and save time-series efficiently (e.g. results for more than 10000 states).

The simulation state is defined by a multi-dimensional index (e.g. time, loadcase and time or time and frequency etc.). This multi-dimensional index is optional for all block types containing the index definition excluding the multi-state blocks. A block without index (nindex = 0) is treated as a constant data block. The data of indexed blocks are only temporarily valid for the defined state, data of constant blocks are globally valid for the whole model and for all simulation states.

Table: Block types and properties

| block | Function *erf{ _read_ \| _write_ }* | mandat. optional | unique repeat. | c = constant s = single-s. m = multi-s. | index mandat. optional | Refers to variables |
|---|---|---|---|---|---|---|
| ERF Header Block | *fileheader* | m | u | c | n/a | no |
| 10 – System Block | *system* | m | u | c | n/a | no |
| 20 – Index Definition Block | *index* | o | u | c | n/a | no |
| 30 – Variable Definition Block | *variables* | o | r | c | n/a | no |
| 40 – Text Block | *text* | o | r | c | n/a | no |
| 100 – Parts Block | *parts* | o | r | c \| s | o | no |
| 200 – Attributes Block | *attributes* | o | r | c \| s | o | no |
| 300 – Element Connectivity Block | *connectivities* | o | r | c \| s | o | no |
| 400 – Collector Block | *collector* | o | r | c \| s | o | no |
| 10xx – Entity Result Block | *entityresults* | o | r | c \| s | o | yes |
| 10xx – Entity Result Block | *multientityresults* | o | r | m | m | yes |
| 11xx – Intra-Elemental Result Block | *elementresults* | o | r | c \| s | o | yes |
| 11xx – Intra-Elemental Result Block | *multielementresults* | o | r | m | m | yes |
| 21xx – Activation Flag Block | *activflags* | o | r | c \| s | o | no |
| 21xx – Activation Flag Block | *multiactivflags* | o | r | m | m | no |
| 10xxx – Matrix Block | *{dense\|triangular\|sparse}matrix* | o | r | c \| s | o | yes |

Table: Examples of user-defined entity types

| etyp | Defined in block | variable | description |
|---|---|---|---|
| NODE | 300 – Element Connectivity Block | etypnode | Nodal point |
| PART | 100 – Parts Block | etyppart | Part |
| RBODY | 400 – Collector Block | etypcoll | Collector of rigid body entities |
| MODEL | 10xx – Entity Result Block | etyp | Global results |
| SHELL | 300 – Element Connectivity Block | etypelem | 4-node shell |
| STRESS | 30 – Variable Definition Block | etypvar | stress tensor $2^{nd}$ rank |
| VELOCITY | 30 – Variable Definition Block | etypvar | velocity vector |

## 2.2 Datatypes 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 | declarations in c | in fortran | application examples |
|---|---|---|---|---|
| `INT` | integer | `int`<br>`long int` | `INTEGER*4` | identifiers, identities, flags |
| `LONG` | integer | `long long int` | `INTEGER*8` | pointers, numbers |
| `FLOAT` | floating point | `float`<br>`double` | `REAL*4`<br>`REAL*8` | results, coefficients, co-ordinates |
| `CHAR[n]` | n characters | `char x[n]` | `CHARACTER(n)` | types, names, expressions |

Note that the precision and size of floating point data is not prescribed in the specification. However, I/O software applications have to make sure that datasets are read or written with the appropriate format. The HDF library provides functionality to define and check the datatype and the size attributes of datasets. HDF also allows 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 each character variable is predefined in the specification (value in squared brackets e.g. CHAR[256]). All character variables have to be padded with space characters (see example below).

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.

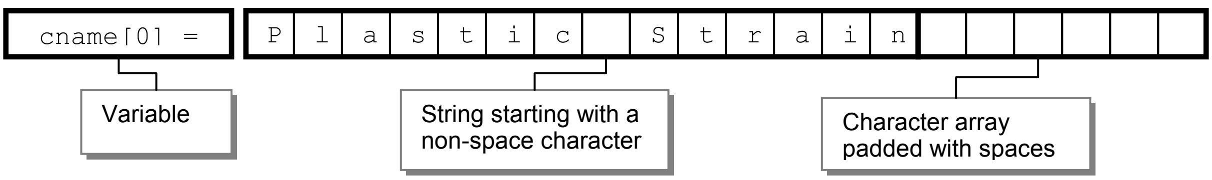



Figure: Example of a character variable

## 2.3 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 scheme below shows an example of a hierarchy of HDF5 groups, attributes and datasets (optional groups in squared brackets [ ] ).

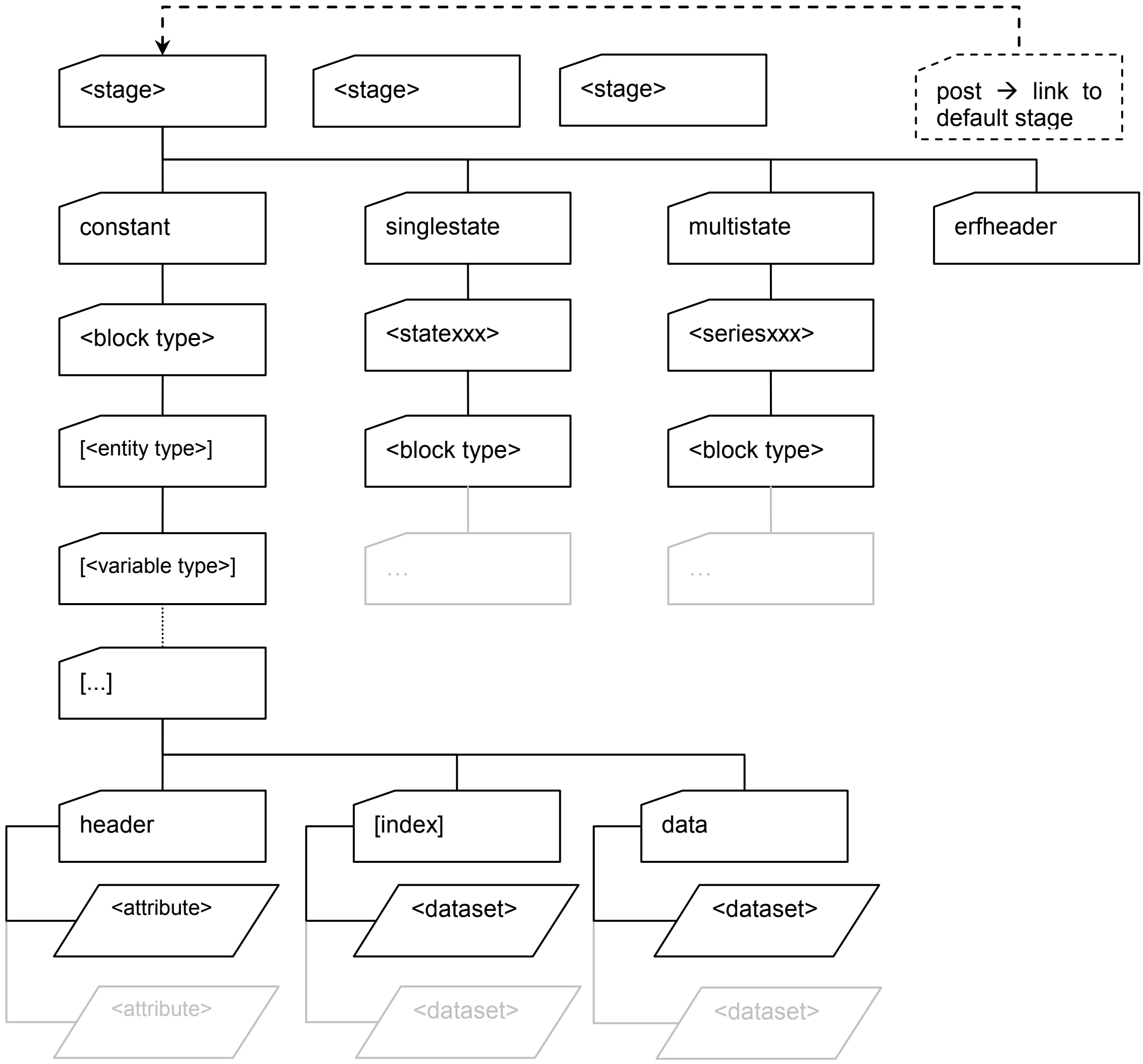



Figure: ERF-HDF5 file storage scheme

The main purpose of the storage scheme is to allow non-interactive reading of data (e.g. for post-processors). Since all data are completely stored in the datasets and attributes respectively, the groups above the block level can be rearranged as needed without loss of information. The figure below shows a snapshot of the HDFView file browser.

ERF-HDF5 datasets are referenced by names (see column “Variable” in 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 and H5S_SIMPLE. H5S_SCALAR defines a scalar variable and H5S_SIMPLE an array. In the block descriptions array variables are indicated by one or more indices, e.g. the variable `x[i][j][k]` is an array of rank 3.

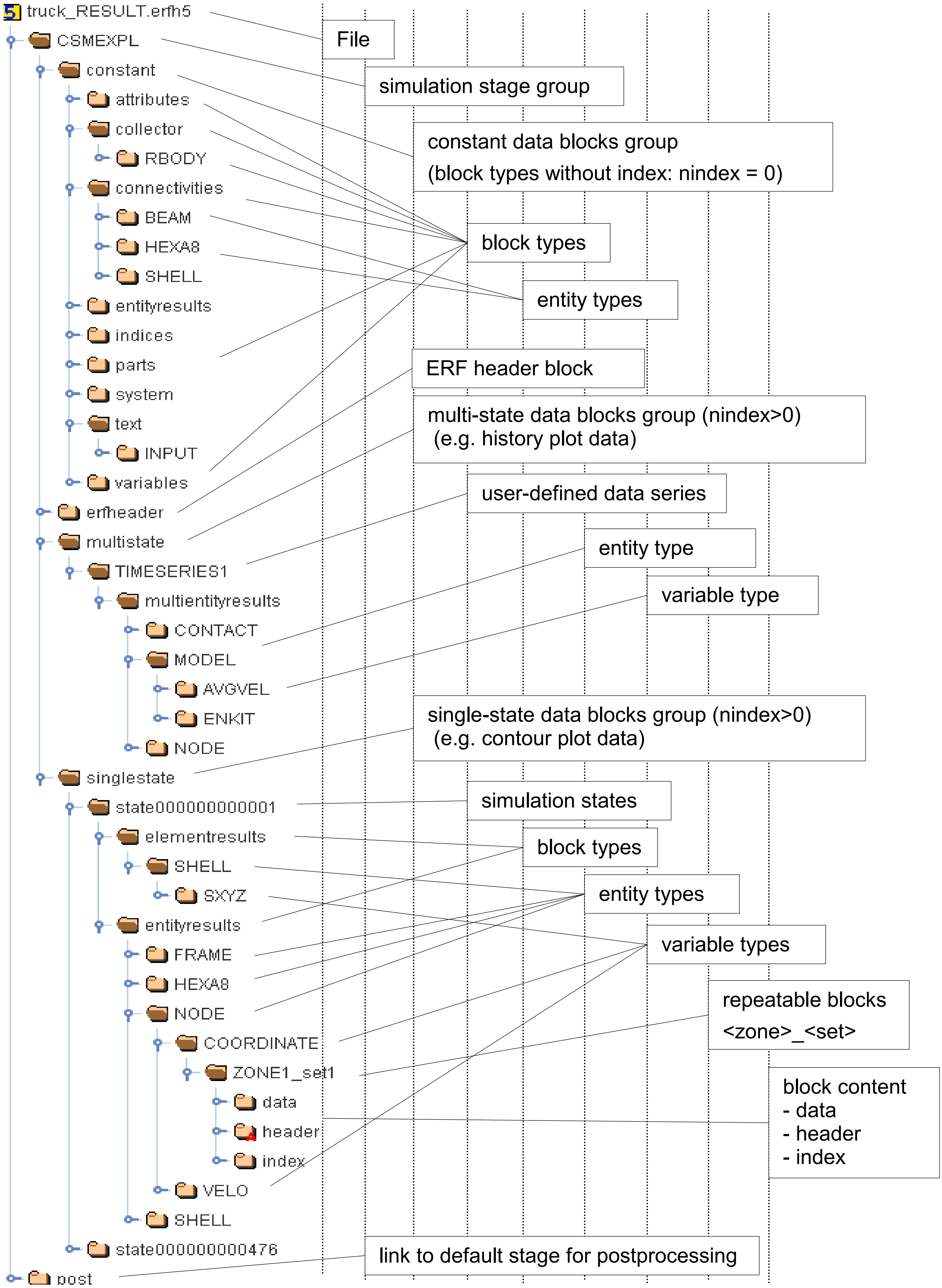


Figure: HDFView snapshot

### 2.4 Simulation State Index

In the ERF format data can be divided into two groups: constant and variable data. Blocks containing variable data are tagged with an index. This index can be single- or multi-dimensional. A typical example of a single-dimensional index is a progression parameter, e.g. time. A multi-dimensional index may consist of two parameters, e.g. loadcase and time.

In ERF the index consists of one or multiple pairs of floating point (indexval) and integer values (indexident). One may, for instance, store the progression parameter (e.g. time) together with the solution increment (see the example below).

The table below shows how the index is stored in the block. Note that for single-state blocks, which contain data of one simulation state, the arrays indexval and indexident have only one subscript, whereas for multi-state blocks, which contain multiple simulation states, these arrays have two subscripts.

The simulation state index must be declared via the index definition block (block-type 20). This index block contains the names of the index parameters as well as their unit dimensions.

Table: Simulation state index

| **Header** | | | |
|---|---|---|---|
| No of indices | nindex | INT | |
| **Index** | **(single-state: optional)** | | |
| single-state block:<br>Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| single-state block:<br>Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| multi-state block:<br>Index identity | (i=0; i<nstate; i++)<br>(j=0; j<nindex; j++)<br>indexident[i][j] | INT | e.g. increment |
| multi-state block:<br>Index Value | (i=0; i<nstate; i++)<br>(j=0; j<nindex; j++)<br>indexval[i][j] | FLOAT | e.g. time,freq,force, … |

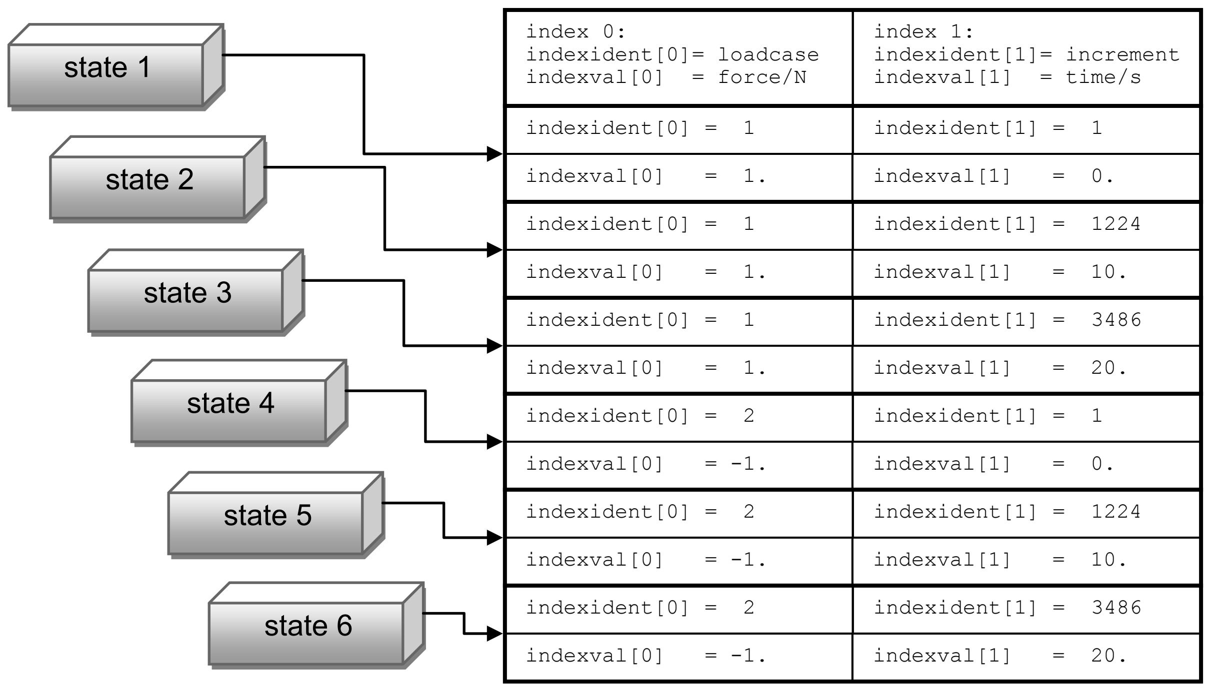



Figure: Example of a 2-dimensional index (loadcase, time).

### 2.5 Frame Definitions

Frames are local or global Cartesian or non-Cartesian coordinate systems. Frames can be referred by any result stored in ERF element- or entityresult blocks. The frame itself is to be stored in an entityresult block. 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

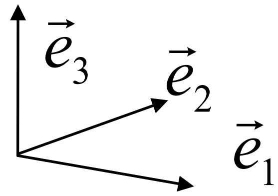



Figure: Cartesian frame

The frame entity type is defined by the parameter etypframe stored in the header of any entity- or elementresult block. Depending on the parameter fswitch, each result can refer to an individual frame fid[i], or all results can refer to the same frame fidglob (see table below).

Table: Frame references in ERF result blocks

| | | | |
|---|---|---|---|
| **Header** | | | |
| Frame type | etypframe | CHAR[256] | e.g. FRAME, FRAME2D |
| Frame control switch | fswitch = { 0 \| 1 } | INT | = 0 → use global fidglob<br>= 1 → apply frame list fid |
| **data** | | | |
| global frame id (optional) | if(fswitch == 0)<br>fidglob | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| list of frame ids (optional) | if(fswitch == 1)<br>(i=0; i<nent; i++)<br>fid[i] | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |

### 2.6 Distributed Memory Processing (DMP)

In order to merge sub-files of computation domains, optional domain decomposition data and reduction operators can be stored in the result blocks. Note that the reduction operations require unique entity identities over all domains.

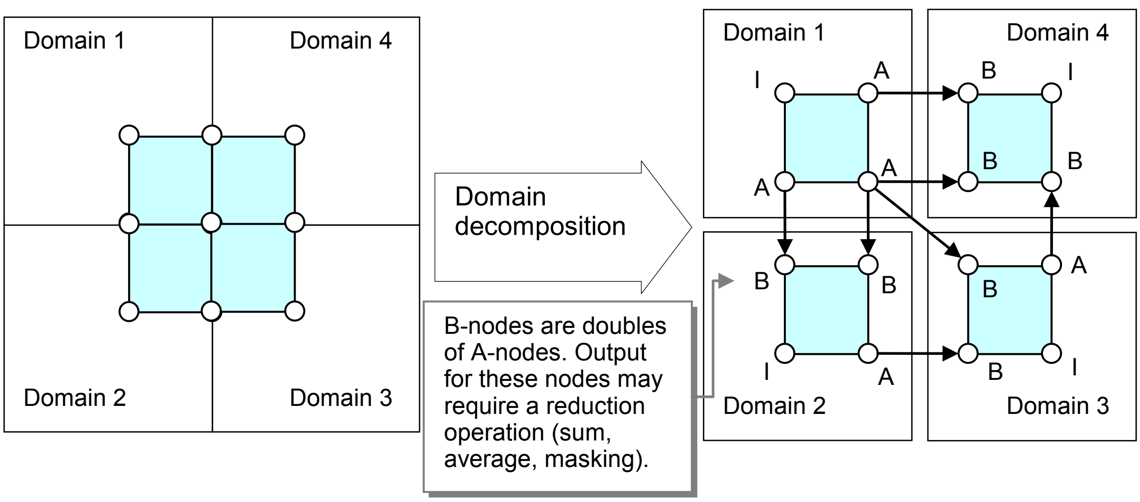



Figure: Example of a DMP domain decomposition scheme for Finite Elements

# 3 ERF Block Definitions

## 3.1 ERF Header Block

*Mandatory, unique*

| Name | Variable [=value] | type | comments |
|---|---|---|---|
| ERF header | erfheader =<br>< 8 chars signature:<br>‘\211’ ‘E’ ‘R’ ‘F’ ‘\r’ ‘\n’ ‘\032’ ‘\n‘ ><br><br>< 32 chars for version numbers:<br>major minor release ><br><br>< 24 chars unused > | CHAR[64] | signature (8 x ASCIIC):<br>Hex: 89 45 52 46 0d 0a 1a<br>Dec: 137 69 82 70 13 10 26 10<br><br>version: three ints separated by spaces:<br>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 – for major specification changes;
2) minor version – for minor specification changes;
3) release version – for software changes.

Example: major=1, minor=2, release=0 => 1 2 0

### 3.2 10 – System Block

*Mandatory, unique*

| Name | Variable [= value] | type | comments |
|---|---|---|---|
| **Header** | | | |
| Block key | block = 10 | INT | |
| Generic block type | blocktype = "system" | CHAR[256] | |
| Model title | title | CHAR[256] | |
| System | sys | CHAR[256] | |
| Solver name | solver_name | CHAR[256] | |
| Solver version | solver_vers | CHAR[256] | |
| Creation Date | cdate | CHAR[8] | YYYYMMDD |
| Creation Time | ctime | CHAR[8] | HHMMSS |
| Modification Date | mdate | CHAR[8] | YYYYMMDD |
| Modification Time | mtime | CHAR[8] | HHMMSS |
| No of base units | nbunit | INT | Can be of arbitrary size.<br>7 SI base units are predefined |
| No. Derived units | ndunit | INT | |
| **Data** | | | |
| Base unit id | (i=0; i<nbunit; i++)<br>ubid[i] | INT | Used to specify:<br>- derived units<br>- data units<br>(see note) |
| Base unit type | (i=0; i<nbunit; i++)<br>ubtyp[i] | CHAR[256] | - Predef. values, see table<br>- other values allowed but cannot be associated with SI units |
| Base unit name | (i=0; i<nbunit; i++)<br>ubnam[i] | CHAR[256] | mm, hour, etc. |
| shift value to SI standard unit | (i=0; i<nbunit; i++)<br>ubshift[i] | FLOAT | allows unit conversions |
| Conversion factor to SI standard unit | (i=0; i<nbunit; i++)<br>ubcon[i] | FLOAT | allows unit conversions |
| Derived unit id | (i=0; i<ndunit; i++)<br>udid[i] | INT | Used to specify data units (see note 1) |
| Derived unit name | (i=0; i<ndunit; i++)<br>udnam[i] | CHAR[256] | e.g. MPa, kN, J |
| Derived unit scale factor | (i=0; i<ndunit; i++)<br>udscal[i] | FLOAT | e.g. $10^6$ |
| No of base units for this derived unit | (i=0; i<ndunit; i++)<br>ndbunit[i] | INT | ndbunit[i]≤ nbunit |
| Base unit id to build this derived unit | (i=0,k=0; i<ndunit; i++)<br>(j=0;j<ndbunit[i];j++,k++)<br>udbid[k] | INT | Defined above |
| Base unit exponents | (i=0,k=0; i<ndunit; i++)<br>(j=0;j<ndbunit[i];j++,k++)<br>udbexp[k] | FLOAT | MPa=$10^6$kg/s²/m<br>Fractional units allowed (fractals) |

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 |

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:

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

with:

| | |
|---|---|
| $U$ | data unit |
| $V_i$ | Component unit (base or derived) |
| $\lambda_i$ , $\mu_i$ | component exponents |

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

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

$$V_{derived\ j} = \alpha_{derived\ j} \prod_i V_{base\ i}^{\nu_i}$$

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 |
| $\nu_i$ | exponent for the base unit to build the derived unit |

Note: $\lambda_i$ , $\mu_i$, $\nu_i$ $\neq 1$ only allowed for non-shifted base units with $S_i = 0$

### 3.3 20 – Index Definition Block

*Optional, unique*

| Name | Variable [= value] | type | comments |
|---|---|---|---|
| **Header** | | | |
| Block key | block = 20 | INT | |
| Generic block type | blocktype = “indices” | CHAR[256] | |
| Number of Indices | nindex | INT | |
| **Data** | | | |
| Index type | (i=0; i<nindex; i++)<br>etypindex[i] | CHAR[256] | Time, freq, loadcase,<br>increment, punchstroke, etc. |
| No of base units | (i=0; i<nindex; i++)<br>mbunit[i] | INT | mbunit[i] ≤ nbunit |
| No derived units | (i=0; i<nindex; i++)<br>mdunit[i] | INT | mdunit[i] ≤ ndunit |
| Base unit id | (i=0,k=0;i<nindex;i++)<br>(j=0;j<mbunit[i];j++,k++)<br>ubid[k] | INT | |
| Base unit exponents | (i=0,k=0;i<nindex;i++)<br>(j=0;j<mbunit[i];j++,k++)<br>ubexp[k] | FLOAT | |
| Derived unit id | (i=0,k=0;i<nindex;i++)<br>(j=0;j<mdunit[i];j++,k++)<br>udid[k] | INT | |
| Derived unit exponents | (i=0,k=0;i<nindex;i++)<br>(j=0;j<mdunit[i];j++,k++)<br>udexp[k] | FLOAT | |

### 3.4 30 – Variable Definition Block

*Optional, unique per etypvar*

| Name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 30 | INT | |
| Generic block type | blocktype = "variables" | CHAR[256] | |
| variable type | etypvar | CHAR[256] | user-defined |
| transformation rule | idtrans | INT | =0: non<br>>0: user-defined |
| rank | rank | INT | Scalar: 0, vector: 1<br>2$^{nd}$ order tensor: 2, … |
| dimension | ndim | INT | n/a for rank = 0 |
| No of coordinates | ncoo | INT | see table |
| **data** | | | |
| No of base units | (i=0; i<ncoo; i++)<br>mbunit[i] | INT | mbunit[i] ≤ nbunit |
| No derived units | (i=0; i<ncoo; i++)<br>mdunit[i] | INT | mdunit[i] ≤ ndunit |
| Base unit id | (i=0,k=0;i<ncoo;i++)<br>(j=0;j<mbunit[i];j++,k++)<br>ubid[k] | INT | |
| Base unit exponents | (i=0,k=0;i<ncoo;i++)<br>(j=0;j<mbunit[i];j++,k++)<br>ubexp[k] | FLOAT | |
| Derived unit id | (i=0,k=0;i<ncoo;i++)<br>(j=0;j<mdunit[i];j++,k++)<br>udid[k] | INT | |
| Derived unit exponents | (i=0,k=0;i<ncoo;i++)<br>(j=0;j<mdunit[i];j++,k++)<br>udexp[k] | FLOAT | |
| coordinate name | (i=0; i<ncoo; i++)<br>cname[i] | CHAR[256] | self-explanatory name e.g. Stress_xx, Stress_yy, etc. |
| coordinate id | (i=0; i<ncoo; i++)<br>cid[i] | INT | |
| | (i=0;i<ndim;i++){ | | loop 0 |
| | (j=0;j<ndim;j++){ | | loop 1 |
| | … | | … |
| | (n=0;n<ndim;n++){ | | loop rank-1 |
| cid of component | comp[i][j][…][n] | INT | - components = ndim$^{rank}$<br>- symmetrical components have the same id |
| | } | | |
| | … | | |
| | } | | |
| | } | | |

<u>Examples of variable definitions</u>:

| variable | type | definition | rank | ncoo | ndim | no of comp. | Coordinate id |
|---|---|---|---|---|---|---|---|
| mass | scalar | m | 0 | 1 | n/a | n/a | 1 |
| velocity | vector | $\vec{v} = v_i \vec{e}_i$ | 1 | 3 | 3 | 3 | 1 2 3 |
| stress | symmetric tensor | $\vec{\vec{\sigma}} = \sigma_{ij} \vec{e}_i \vec{e}_j$ | 2 | 6 | 3 | 9 | 1 2 4<br>2 3 5<br>4 5 6 |
| material tensor | asymmetric tensor | $\vec{C}^4 = C_{ijkl} \vec{e}_i \vec{e}_j \vec{e}_k \vec{e}_l$ | 4 | 81 | 3 | 81 | 1 4 7<br>2 5 8<br>3 6 9 |
| damage | array of scalars | $\underline{D} = (d_1, d_2, d_3)^T$ | 0 | 3 | n/a | n/a | 1 2 3 |

### 3.5 40 – Text Block

*Optional, unique per etyptext*

| Name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 40 | INT | |
| Generic block type | blocktype = "text" | CHAR[256] | |
| Textblock type | etyptext | CHAR[256] | user-defined, e.g. INPUT |
| Number of lines | nlines | INT | |
| Line length | length | INT | |
| **data** | | | |
| Text | (i=0; i<nlines; i++) line[i] | CHAR[length] | |

The following text

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

may be stored as multiple lines with fix length:

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| line[0] = | T | h | i | s | | i | s | | a | | s | a | m | p | l | e | | | | |
| line[1] = | t | e | x | t | | t | o | | d | e | m | o | n | s | t | r | a | t | e | |
| line[2] = | t | e | x | t | b | l | o | c | k | s | . | | | | | | | | | |

Alternatively, the text can be stored more compactly as a single line using C-style escape sequences, e.g. as end-of-line characters:

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| line[0] = | T | h | i | s | | i | s | | a | | s | a | m | p | l | e | \n | | | |
| | t | e | x | t | | t | o | | d | e | m | o | n | s | t | r | a | t | e | \n |
| | t | e | x | t | b | l | o | c | k | s | . | | | | | | | | | |

### 3.6 100 – Parts Block

*Optional, unique per etyppart and index*

| Name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 100 | INT | |
| Generic block type | blocktype = “parts” | CHAR[256] | |
| part entity type | etyppart | CHAR[256] | user-defined |
| Number of parts | npart | INT | |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++) indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++) indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| Part id | (i=0; i<npart; i++) pid[i] | INT | |
| Part title | (i=0; i<npart; i++) title[i] | CHAR[256] | |
| Material Id | (i=0; i<npart; i++) mid[i] | INT | |
| Material type id | (i=0; i<npart; i++) mtyp[i] | INT | |
| Color identifier | (i=0; i<npart; i++) pcol[i] | INT | Predefined, table |
| Viewtype identifier | (i=0; i<npart; i++) pvtyp[i] | INT | Predefined, table |

Table – color definitions (example):

| Color id | Color name | R | G | B |
|---|---|---|---|---|
| 1 | black | 0 | 0 | 0 |
| 2 | white | 255 | 255 | 255 |
| 3 | red | 255 | 0 | 0 |
| 4 | green | 0 | 255 | 0 |
| 5 | blue | 0 | 0 | 255 |
| 6 | yellow | 255 | 255 | 0 |

Table – view types (example):

| viewtype id | viewtype name |
|---|---|
| 1 | wireframe |
| 2 | shaded |
| 3 | shaded + edges |
| 4 | transparent shaded |
| 5 | transparent shaded + edges |

### 3.7 200 – Attributes Block

*Optional, unique per etyp and index*

| Name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 200 | INT | |
| Generic block type | blocktype = “attributes” | CHAR[256] | |
| Entity type | etyp | CHAR[256] | e.g. PART, SHELL, NODE, … |
| Number of entities | nent | INT | |
| Number of user ids | nuid | INT | |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| Entity ID | (i=0; i<nent; i++)<br>entid[i] | INT | |
| Entity title | (i=0; i<nent; i++)<br>title[i] | CHAR[256] | |
| User ID j of ent. i | (i=0; i<nent; i++)<br>(j=0; j<nuid; j++)<br>uid[i][j] | INT | also for color or viewtype |

### 3.8 300 – Element Connectivity Block

*optional, unique per etypelem and index*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 300 | INT | |
| Generic block type | blocktype="connectivities" | CHAR[256] | |
| element entity type | etypelem | CHAR[256] | user-defined |
| part entity type | etyppart | CHAR[256] | user-defined |
| node entity type | etypnode | CHAR[256] | user-defined |
| No of elements | nele | INT | |
| No of nodes | npele | INT | |
| element dimensions | ndim | INT | point=0, bar=1, shell=2, solid=3 |
| No of int param. | nbint | INT | |
| No of float param. | nbfloat | INT | with dimensions below |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| Element id | (i=0; i<nele; i++)<br>idele[i] | INT | |
| Part Id | (i=0; i<nele; i++)<br>pid[i] | INT | |
| Node connectivity | (i=0; i<nele; i++)<br>(j=0; j<npele; j++)<br>ic[i][j] | INT | |
| No of base units | (i=0; i<nbfloat; i++)<br>mbunit[i] | INT | mbunit[i] ≤ nbunit |
| No derived units | (i=0; i<nbfloat; i++)<br>mdunit[i] | INT | mdunit[i] ≤ ndunit |
| Base unit id | (i=0,k=0;i<nbfloat;i++)<br>(j=0;j<mbunit[i];j++,k++)<br>ubid[k] | INT | |
| Base unit exponents | (i=0,k=0;i<nbfloat;i++)<br>(j=0;j<mbunit[i];j++,k++)<br>ubexp[k] | FLOAT | |
| Derived unit id | (i=0,k=0;i<nbfloat;i++)<br>(j=0;j<mdunit[i];j++,k++)<br>udid[k] | INT | |
| Derived unit exponents | (i=0,k=0;i<nbfloat;i++)<br>(j=0;j<mdunit[i];j++,k++)<br>udexp[k] | FLOAT | |
| int params | (i=0; i<nele; i++)<br>(j=0; j<nbint; j++)<br>iparam[i][j] | INT | |
| float params | (i=0; i<nele; i++)<br>(j=0; j<nbfloat; j++)<br>fparam[i][j] | FLOAT | |

### 3.9 400 – Collector Block

*Optional, unique per etypcoll and index*

| name | Variable [= value] | type | Comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 400 | INT | |
| Generic block type | blocktype = “collector” | CHAR[256] | |
| collector entity type | etypcoll | CHAR[256] | user-defined |
| no of Collectors | ncoll | INT | |
| No of entity types | ntyp | INT | |
| Total no of entities | nenttot | INT | = SUM(ntyp*ncoll)nent[i] |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| Collector id | (i=0; i<ncoll; i++)<br>idcoll[i] | INT | |
| Entity type | (i=0; i<ntyp; i++)<br>etyp[i] | CHAR[256] | e.g. NODE, SHELL, … |
| No of entities per collector | (i=0; i<ncoll; i++)<br>(j=0; j<ntyp; j++)<br>nent[i][j] | INT | |
| entity id | (i=0,n=0;i<ncoll;i++)<br>(j=0;j<ntyp;j++)<br>(k=0;k<nent[i][j];k++,n++)<br>entid[n] | INT | |

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

## 3.10 10xx – Entity Result Block

### 3.10.1 1000 – Entity Result Block for Data of Type Real

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 1000 | INT | |
| Generic block type | blocktype="entityresults" | CHAR[256] | |
| entity type | etyp | CHAR[256] | user-defined |
| No of entities | nent | INT | |
| Variable type | etypvar | CHAR[256] | → variable block |
| no of coordinates | ncoo | INT | → variable block |
| Zone type | etypzone | CHAR[256] | e.g. NONE, PLY, LAYER |
| Zone id | zoneid | INT | → unique id of etypzone |
| Frame type | etypframe | CHAR[256] | e.g. FRAME, FRAME2D |
| Frame control switch | fswitch = { 0 \| 1 } | INT | = 0 → use global fidglob<br>= 1 → apply frame list fid |
| DMP option switch: | dmpswitch = { 0 \| 1 } | INT | 0: off, 1: on |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| entity id | (i=0; i<nent; i++)<br>entid[i] | INT | |
| global frame id (optional) | if(fswitch == 0)<br>fidglob | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| list of frame ids (optional) | if(fswitch == 1)<br>(i=0; i<nent; i++)<br>fid[i] | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| DMP merge operators (optional) | if(dmpswitch == 1)<br>(i=0; i<ncoo; i++)<br>operator[i] | CHAR[256] | e.g. SUM, OR, MAX |
| DMP domain weights (optional) | if(dmpswitch == 1)<br>(i=0; i<nent; i++)<br>weight[i] | FLOAT | |
| Result coordinate | (i=0; i<nent; i++)<br>(j=0; j<ncoo; j++)<br>res[i][j] | FLOAT | |

### 3.10.2 1001 – Entity Result Block for Data of Type Complex

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 1001 | INT | |
| Generic block type | blocktype="entityresults" | CHAR[256] | |
| entity type | etyp | CHAR[256] | user-defined |
| No of entities | nent | INT | |
| Variable type | etypvar | CHAR[256] | → variable block |
| no of coordinates | ncoo | INT | → variable block |
| Zone type | etypzone | CHAR[256] | e.g. NONE, PLY, LAYER |
| Zone id | zoneid | INT | → unique id of etypzone |
| Frame type | etypframe | CHAR[256] | e.g. FRAME, FRAME2D |
| Frame control switch | fswitch = { 0 \| 1 } | INT | = 0 → use global fidglob<br>= 1 → apply frame list fid |
| DMP option switch: | dmpswitch = { 0 \| 1 } | INT | 0: off, 1: on |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| entity id | (i=0; i<nent; i++)<br>entid[i] | INT | |
| global frame id (optional) | if(fswitch == 0)<br>fidglob | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| list of frame ids (optional) | if(fswitch == 1)<br>(i=0; i<nent; i++)<br>fid[i] | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| DMP merge operators (optional) | if(dmpswitch == 1)<br>(i=0; i<ncoo; i++)<br>operator[i] | CHAR[256] | e.g. SUM, OR, MAX |
| DMP domain weights (optional) | if(dmpswitch == 1)<br>(i=0; i<nent; i++)<br>weight[i] | FLOAT | |
| Result coordinate | (i=0; i<nent; i++)<br>(j=0; j<ncoo; j++)<br>(k=0; k<2; k++)<br>res[i][j][k] | FLOAT | k=0: real part<br>k=1: imaginary part |

### 3.10.3 1050 – Multi-State Entity Result Block for Data of Type Real

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 1050 | INT | |
| Generic block type | blocktype="multientityresults" | CHAR[256] | |
| series name | series | CHAR[256] | user-defined |
| entity type | etyp | CHAR[256] | user-defined |
| No of entities | nent | INT | |
| Variable type | etypvar | CHAR[256] | → variable block |
| no of coordinates | ncoo | INT | → variable block |
| Zone type | etypzone | CHAR[256] | e.g. NONE, PLY, LAYER |
| Zone id | zoneid | INT | → unique id of etypzone |
| Frame type | etypframe | CHAR[256] | e.g. FRAME, FRAME2D |
| Frame control switch | fswitch = { 0 \| 1 } | INT | = 0 → use global fidglob<br>= 1 → apply frame list fid |
| DMP option switch: | dmpswitch = { 0 \| 1 } | INT | 0: off, 1: on |
| No of states | nstate | INT | |
| No of indices | nindex | INT | |
| **index (mandatory)** | | | |
| Index identity | (i=0; i<nstate; i++)<br>(j=0; j<nindex; j++)<br>indexident[i][j] | INT | e.g. increment |
| Index Value | (i=0; i<nstate; i++)<br>(j=0; j<nindex; j++)<br>indexval[i][j] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| entity id | (i=0; i<nent; i++)<br>entid[i] | INT | |
| global frame id (optional) | if(fswitch == 0)<br>fidglob | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| list of frame ids (optional) | if(fswitch == 1)<br>(i=0; i<nent; i++)<br>fid[i] | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| DMP merge operators (optional) | if(dmpswitch == 1)<br>(i=0; i<ncoo; i++)<br>operator[i] | CHAR[256] | e.g. SUM, OR, MAX |
| DMP domain weights (optional) | if(dmpswitch == 1)<br>(i=0; i<nent; i++)<br>weight[i] | FLOAT | |
| Result coordinate | (i=0; i<nstate; i++)<br>(j=0; j<nent; j++)<br>(k=0; k<ncoo; k++)<br>res[i][j][k] | FLOAT | |

### 3.10.4 1051 – Multi-State Entity Result Block for Data of Type Complex

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 1051 | INT | |
| Generic block type | blocktype="multientityresults" | CHAR[256] | |
| series name | series | CHAR[256] | user-defined |
| entity type | etyp | CHAR[256] | user-defined |
| No of entities | nent | INT | |
| Variable type | etypvar | CHAR[256] | → variable block |
| no of coordinates | ncoo | INT | → variable block |
| Zone type | etypzone | CHAR[256] | e.g. NONE, PLY, LAYER |
| Zone id | zoneid | INT | → unique id of etypzone |
| Frame type | etypframe | CHAR[256] | e.g. FRAME, FRAME2D |
| Frame control switch | fswitch = { 0 \| 1 } | INT | = 0 → use global fidglob<br>= 1 → apply frame list fid |
| DMP option switch: | dmpswitch = { 0 \| 1 } | INT | 0: off, 1: on |
| No of states | nstate | INT | |
| No of indices | nindex | INT | |
| **index (mandatory)** | | | |
| Index identity | (i=0; i<nstate; i++)<br>(j=0; j<nindex; j++)<br>indexident[i][j] | INT | e.g. increment |
| Index Value | (i=0; i<nstate; i++)<br>(j=0; j<nindex; j++)<br>indexval[i][j] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| entity id | (i=0; i<nent; i++)<br>entid[i] | INT | |
| global frame id (optional) | if(fswitch == 0)<br>fidglob | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| list of frame ids (optional) | if(fswitch == 1)<br>(i=0; i<nent; i++)<br>fid[i] | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| DMP merge operators (optional) | if(dmpswitch == 1)<br>(i=0; i<ncoo; i++)<br>operator[i] | CHAR[256] | e.g. SUM, OR, MAX |
| DMP domain weights (optional) | if(dmpswitch == 1)<br>(i=0; i<nent; i++)<br>weight[i] | FLOAT | |
| Result coordinate | (i=0; i<nstate; i++)<br>(j=0; j<nent; j++)<br>(k=0; k<ncoo; k++)<br>(l=0; l<2; l++)<br>res[i][j][k][l] | FLOAT | l=0: real part<br>l=1: imaginary part |

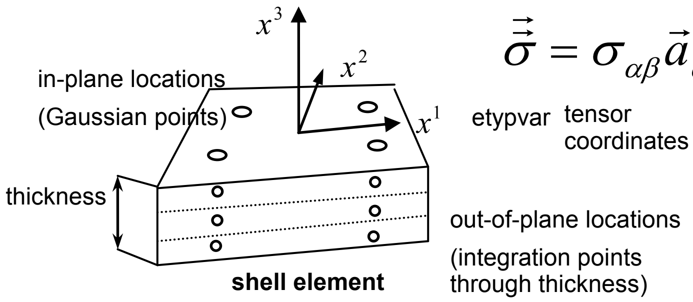
## 3.11 11xx – Intra-Elemental Result Block

### 3.11.1 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).

<u>**Example:** shell element, output of a symmetric plane stress tensor</u>

- 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

| (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 |

### 3.11.2 1100 – Intra-Elemental Result Block for Data of Type Real

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 1100 | INT | |
| Generic block type | blocktype="elementresults" | CHAR[256] | |
| element entity type | etypelem | CHAR[256] | user-defined |
| No of elements | nele | INT | |
| Variable type | etypvar | CHAR[256] | → variable block |
| no of coordinates | ncoo | INT | → variable block |
| no of intra-locations | numl | INT | |
| intra-location dims. | ndim | INT | |
| Zone type | etypzone | CHAR[256] | e.g. NONE, PLY, LAYER |
| Zone id | zoneid | INT | → unique id of etypzone |
| Frame type | etypframe | CHAR[256] | e.g. FRAME, FRAME2D |
| Frame control switch | fswitch = { 0 \| 1 \| 2 } | INT | = 0 → use global fidglob<br>= 1 → use list fid per element<br>= 2 → fid per intra-location |
| DMP option switch: | dmpswitch = { 0 \| 1 } | INT | 0: off, 1: on |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| element id | (i=0; i<nele; i++)<br>idele[i] | INT | → connectivity block |
| global frame id (optional) | if(fswitch == 0)<br>fidglob | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| list of frame ids (optional) | if(fswitch == 1)<br>(i=0; i<nele; i++)<br>fid[i]<br><br>if(fswitch == 2)<br>(i=0; i<nele; i++)<br>(j=0; j<numl; j++)<br>fid[i][j] | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| DMP merge operators (optional) | if(dmpswitch == 1)<br>(i=0; i<ncoo; i++)<br>operator[i] | CHAR[256] | e.g. SUM, OR, MAX |
| DMP domain weights (optional) | if(dmpswitch == 1)<br>(i=0; i<nele; i++)<br>weight[i] | FLOAT | |
| Isoparametric coordinates of intra-locations | (i=0; i<numl; i++)<br>(j=0; j<ndim; j++)<br>eloc[i][j] | FLOAT | |
| Result coordinate | (i=0; i<nele; i++)<br>(j=0; j<numl; j++)<br>(k=0; k<ncoo; k++)<br>res[i][j][k] | FLOAT | |

### 3.11.3 1101 – Intra-Elemental Result Block for Data of Type Complex

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 1101 | INT | |
| Generic block type | blocktype="elementresults" | CHAR[256] | |
| element entity type | etypelem | CHAR[256] | user-defined |
| No of elements | nele | INT | |
| Variable type | etypvar | CHAR[256] | → variable block |
| no of coordinates | ncoo | INT | → variable block |
| no of intra-locations | numl | INT | |
| intra-location dims. | ndim | INT | |
| Zone type | etypzone | CHAR[256] | e.g. NONE, PLY, LAYER |
| Zone id | zoneid | INT | → unique id of etypzone |
| Frame type | etypframe | CHAR[256] | e.g. FRAME, FRAME2D |
| Frame control switch | fswitch = { 0 \| 1 \| 2 } | INT | = 0 → use global fidglob<br>= 1 → use list fid per element<br>= 2 → fid per intra-location |
| DMP option switch: | dmpswitch = { 0 \| 1 } | INT | 0: off, 1: on |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| element id | (i=0; i<nele; i++)<br>idele[i] | INT | → connectivity block |
| global frame id (optional) | if(fswitch == 0)<br>fidglob | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| list of frame ids (optional) | if(fswitch == 1)<br>(i=0; i<nele; i++)<br>fid[i]<br><br>if(fswitch == 2)<br>(i=0; i<nele; i++)<br>(j=0; j<numl; j++)<br>fid[i][j] | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| DMP merge operators (optional) | if(dmpswitch == 1)<br>(i=0; i<ncoo; i++)<br>operator[i] | CHAR[256] | e.g. SUM, OR, MAX |
| DMP domain weights (optional) | if(dmpswitch == 1)<br>(i=0; i<nele; i++)<br>weight[i] | FLOAT | |
| Isoparametric coordinates of intra-locations | (i=0; i<numl; i++)<br>(j=0; j<ndim; j++)<br>eloc[i][j] | FLOAT | |
| Result coordinate | (i=0; i<nele; i++)<br>(j=0; j<numl; j++)<br>(k=0; k<ncoo; k++)<br>(l=0; l<2; l++)<br>res[i][j][k][l] | FLOAT | l=0: real part<br>l=1: imaginary part |

### 3.11.4 1150 – Multi-State Intra-Elemental Result Block for Data of Type Real

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 1150 | INT | |
| Generic block type | blocktype=“multielementresults” | CHAR[256] | |
| series name | series | CHAR[256] | user-defined |
| element entity type | etypelem | CHAR[256] | user-defined |
| No of elements | nele | INT | |
| Variable type | etypvar | CHAR[256] | → variable block |
| no of coordinates | ncoo | INT | → variable block |
| no of intra-locations | numl | INT | |
| intra-location dims. | ndim | INT | |
| Zone type | etypzone | CHAR[256] | e.g. NONE, PLY, LAYER |
| Zone id | zoneid | INT | → unique id of etypzone |
| Frame type | etypframe | CHAR[256] | e.g. FRAME, FRAME2D |
| Frame control switch | fswitch = { 0 \| 1 \| 2 } | INT | = 0 → use global fidglob<br>= 1 → use list fid per element<br>= 2 → fid per intra-location |
| DMP option switch: | dmpswitch = { 0 \| 1 } | INT | 0: off, 1: on |
| No of states | nstate | INT | |
| No of indices | nindex | INT | |
| **index (mandatory)** | | | |
| Index identity | (i=0; i<nstate; i++)<br>(j=0; j<nindex; j++)<br>indexident[i][j] | INT | e.g. increment |
| Index Value | (i=0; i<nstate; i++)<br>(j=0; j<nindex; j++)<br>indexval[i][j] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| element id | (i=0; i<nele; i++)<br>idele[i] | INT | → connectivity block |
| global frame id (optional) | if(fswitch == 0)<br>fidglob | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| list of frame ids (optional) | if(fswitch == 1)<br>(i=0; i<nele; i++)<br>fid[i]<br><br>if(fswitch == 2)<br>(i=0; i<nele; i++)<br>(j=0; j<numl; j++)<br>fid[i][j] | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| DMP merge operators (optional) | if(dmpswitch == 1)<br>(i=0; i<ncoo; i++)<br>operator[i] | CHAR[256] | e.g. SUM, OR, MAX |
| DMP domain weights (optional) | if(dmpswitch == 1)<br>(i=0; i<nele; i++)<br>weight[i] | FLOAT | |
| Isoparametric coordinates of intra-locations | (i=0; i<numl; i++)<br>(j=0; j<ndim; j++)<br>eloc[i][j] | FLOAT | |
| Result coordinate | (i=0; i<nstate; i++)<br>(j=0; j<nele; j++)<br>(k=0; k<numl; k++)<br>(l=0; l<ncoo; l++)<br>res[i][j][k][l] | FLOAT | |

### 3.11.5 1151 – Multi-State Intra-Elemental Result Block for Data of Type Complex

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 1151 | INT | |
| Generic block type | blocktype="multielementresults" | CHAR[256] | |
| series name | series | CHAR[256] | user-defined |
| element entity type | etypelem | CHAR[256] | user-defined |
| No of elements | nele | INT | |
| Variable type | etypvar | CHAR[256] | → variable block |
| no of coordinates | ncoo | INT | → variable block |
| no of intra-locations | numl | INT | |
| intra-location dims. | ndim | INT | |
| Zone type | etypzone | CHAR[256] | e.g. NONE, PLY, LAYER |
| Zone id | zoneid | INT | → unique id of etypzone |
| Frame type | etypframe | CHAR[256] | e.g. FRAME, FRAME2D |
| Frame control switch | fswitch = { 0 \| 1 \| 2 } | INT | = 0 → use global fidglob<br>= 1 → use list fid per element<br>= 2 → fid per intra-location |
| DMP option switch: | dmpswitch = { 0 \| 1 } | INT | 0: off, 1: on |
| No of states | nstate | INT | |
| No of indices | nindex | INT | |
| **index (mandatory)** | | | |
| Index identity | (i=0; i<nstate; i++)<br>(j=0; j<nindex; j++)<br>indexident[i][j] | INT | e.g. increment |
| Index Value | (i=0; i<nstate; i++)<br>(j=0; j<nindex; j++)<br>indexval[i][j] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| element id | (i=0; i<nele; i++)<br>idele[i] | INT | → connectivity block |
| global frame id<br>(optional) | if(fswitch == 0)<br>fidglob | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| list of frame ids<br>(optional) | if(fswitch == 1)<br>(i=0; i<nele; i++)<br>fid[i]<br><br>if(fswitch == 2)<br>(i=0; i<nele; i++)<br>(j=0; j<numl; j++)<br>fid[i][j] | INT | = 0 → global cartesian<br>> 0 → frame id<br>< 0 → local element frame |
| DMP merge operators<br>(optional) | if(dmpswitch == 1)<br>(i=0; i<ncoo; i++)<br>operator[i] | CHAR[256] | e.g. SUM, OR, MAX |
| DMP domain weights<br>(optional) | if(dmpswitch == 1)<br>(i=0; i<nele; i++)<br>weight[i] | FLOAT | |
| Isoparametric coordinates of intra-locations | (i=0; i<numl; i++)<br>(j=0; j<ndim; j++)<br>eloc[i][j] | FLOAT | |
| Result coordinate | (i=0; i<nstate; i++)<br>(j=0; j<nele; j++)<br>(k=0; k<numl; k++)<br>(l=0; l<ncoo; l++)<br>(m=0; m<2; m++)<br>res[i][j][k][l][m] | FLOAT | m=0: real part<br>m=1: imaginary part |

## 3.12 21xx – Activation Flag Block

### 3.12.1 2100 - Activation Flag Block

*Optional, unique per etyp and index*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 2100 | INT | |
| Generic block type | blocktype="activflags" | CHAR[256] | |
| entity type | etyp | CHAR[256] | user-defined |
| No of flagged entities | nent | INT | |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++) indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++) indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| entity id | (i=0; i<nent; i++) entid[i] | INT | |
| Flag status | (i=0; i<nent; i++) status[i] | INT | |

### 3.12.2 2150 – Multi-State Activation Flag Block

*Optional, unique per etyp and index*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 2150 | INT | |
| Generic block type | blocktype="multiactivflags" | CHAR[256] | |
| series name | series | CHAR[256] | user-defined |
| entity type | etyp | CHAR[256] | user-defined; see table "entity type" |
| No of flagged entities | nent | INT | |
| No of states | nstate | INT | |
| No of indices | nindex | INT | |
| **index (mandatory)** | | | |
| Index identity | (i=0; i<nstate; i++) (j=0; j<nindex; j++) indexident[i][j] | INT | e.g. increment |
| Index Value | (i=0; i<nstate; i++) (j=0; j<nindex; j++) indexval[i][j] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| entity id | (i=0; i<nent; i++) entid[i] | INT | |
| Flag status | (i=0; i<nstate; i++) (j=0; j<nent; j++) status[i][j] | INT | |

## 3.13 10xxx – Matrix Block

### 3.13.1 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:

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

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

Sparse matrix CRS (Compressed Sparse Row) Storage Scheme:

$$val(k) = a_{ij}\ ,\quad i,\ j = 1 \ldots nrow\ ,\quad k = 1 \ldots nnz$$
$$colind(k) = j\ ,\quad rowptr(i) \le k < rowptr(i+1)\ ,\quad rowptr(nrow+1) = nnz + 1$$

### 3.13.2 10000 – Dense Matrix Block for Data of Type Real

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 10000 | INT | |
| Generic block type | blocktype="densematrix" | CHAR[256] | |
| entity type | etyp | CHAR[256] | user-defined |
| entity id | entid | INT | |
| variable type | etypvar | CHAR[256] | |
| No of rows | nrow | INT | |
| No of columns | ncolumn | INT | |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| matrix values | (i=0; i<nrow; i++)<br>(j=0; j<ncolumn; j++)<br>val[i][j] | FLOAT | |

### 3.13.3 10001 – Dense Matrix for Data of Type Complex

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 10001 | INT | |
| Generic block type | blocktype="densematrix" | CHAR[256] | |
| entity type | etyp | CHAR[256] | user-defined |
| entity id | entid | INT | |
| variable type | etypvar | CHAR[256] | |
| No of rows | nrow | INT | |
| No of columns | ncolumn | INT | |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| matrix values | (i=0; i<nrow; i++)<br>(j=0; j<ncolumn; j++)<br>(k=0; k<2; k++)<br>val[i][j][k] | FLOAT | k=0: real part<br>k=1: imaginary part |

### 3.13.4 10100 – Triangular and Symmetric Matrix Block for Data of Type Real

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 10100 | INT | |
| Generic block type | blocktype="triangularmatrix" | CHAR[256] | |
| entity type | etyp | CHAR[256] | user-defined |
| entity id | entid | INT | |
| variable type | etypvar | CHAR[256] | |
| No of rows or columns | nsize | INT | |
| Matrix type | matrixtype | INT | see table above |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++) indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++) indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| matrix values | (i=0; i<nsize*(nsize+1)/2; i++) val[i] | FLOAT | |

### 3.13.5 10101 – Triangular and Symmetric Matrix Block for Data of Type Complex

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 10101 | INT | |
| Generic block type | blocktype="triangularmatrix" | CHAR[256] | |
| entity type | etyp | CHAR[256] | user-defined |
| entity id | entid | INT | |
| variable type | etypvar | CHAR[256] | |
| No of rows or columns | nsize | INT | |
| Matrix type | matrixtype | INT | see table above |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++) indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++) indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| matrix values | (i=0; i<nsize*(nsize+1)/2; i++) (j=0; j<2; j++) val[i][j] | FLOAT | j=0: real part<br>j=1: imaginary part |

### 3.13.6 10200 – Sparse CRS Matrix Block for Data of Type Real

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 10200 | INT | |
| Generic block type | blocktype="sparsematrix" | CHAR[256] | |
| entity type | etyp | CHAR[256] | user-defined |
| entity id | entid | INT | |
| variable type | etypvar | CHAR[256] | |
| No of rows = columns | nrow | INT | |
| No of non-zero values | nnz | LONG | |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| column index | (i=0; i<nnz; i++)<br>colind[i] | INT | |
| row pointer | (i=0; i<nrow+1; i++)<br>rowptr[i] | LONG | |
| non-zero matrix values | (i=0; i<nnz; i++)<br>val[i] | FLOAT | |

### 3.13.7 10201 – Sparse CRS Matrix Block for Data of Type Complex

*Optional, repeatable*

| name | Variable [= value] | type | comments |
|---|---|---|---|
| **header** | | | |
| Block key | block = 10201 | INT | |
| Generic block type | blocktype="sparsematrix" | CHAR[256] | |
| entity type | etyp | CHAR[256] | user-defined |
| entity id | entid | INT | |
| variable type | etypvar | CHAR[256] | |
| No of rows = columns | nrow | INT | |
| No of non-zero values | nnz | LONG | |
| No of indices | nindex | INT | |
| **index (optional)** | | | |
| Index identity | (i=0; i<nindex; i++)<br>indexident[i] | INT | e.g. increment |
| Index Value | (i=0; i<nindex; i++)<br>indexval[i] | FLOAT | e.g. time,freq,force, … |
| **data** | | | |
| column index | (i=0; i<nnz; i++)<br>colind[i] | INT | |
| row pointer | (i=0; i<nrow+1; i++)<br>rowptr[i] | LONG | |
| non-zero matrix values | (i=0; i<nnz; i++)<br>(j=0; j<2; j++)<br>val[i][j] | FLOAT | j=0: real part<br>j=1: imaginary part |