SANDIA REPORT

SAND2011-5715
Unlimited Release
Printed August 16, 2011

A Collection of Exodus Utilities: Exodiff, EPU, EJoin, and Conjoin

Gregory D. Sjaardema

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2011-5715
Unlimited Release
Printed August 16, 2011

A Collection of Exodus Utilities: Exodiff, EPU, EJoin, and Conjoin

Gregory D. Sjaardema
Simulation Modeling Sciences Department
Sandia National Laboratories
Albuquerque, NM 87185-0380

Abstract

The applications Exodiff, EPU, EJoin, and Conjoin are members of the Sandia Engineering Analysis Code Access System (SEACAS [5]) which is used by analysts at Sandia National Laboratories. The applications all read and/or write finite element databases stored in the EXODUS [1] format.

Each application is targeted at a few specific areas of functionality:

Exodiff compares the results data from two EXODUS files with user-specified relative or absolute tolerances.

EPU combines multiple EXODUS databases produced by a parallel application into a single EXODUS database.

EJoin joins two or more EXODUS databases into a single EXODUS database. The input databases must have disjoint meta and bulk data.

Conjoin joins two or more EXODUS databases into a single database. The input databases should represent the same model geometry with results data written sequentially.

Contents

1 Introduction **7**

1.1 EXODUS Concepts . . . 8

1.1.1 Meta Data and Bulk Data . . . 8

1.1.2 Database Sections . . . 9

1.2 Licensing . . . 9

2 Exodiff **11**

2.1 Introduction . . . 11

2.1.1 Difference Terminology . . . 11

2.2 Invoking Exodiff . . . 12

2.2.1 Optional Parameters . . . 13

2.2.2 Exodiff Command File Syntax . . . 16

2.3 Examples . . . 17

3 EPU **24**

3.1 Introduction . . . 24

3.2 Invoking EPU . . . 24

3.2.0.1 File Naming Convention . . . 25

3.2.0.2 Defaults and Requirements . . . 25

3.2.1 Options . . . 25

3.2.1.1 Basic invocation options . . . 25

3.2.1.2 Disk-related filename options . . . 25

3.2.1.3 Additional Options . . . 27

3.3 Example . . . 29

3.4 Related Codes . . . 31

4 EJoin **32**

4.1 Introduction . . . 32
4.2 Invoking EJoin . . . 32
4.2.1 Options . . . 32
4.3 Examples . . . 34
4.4 Related Codes . . . 35

5 Conjoin **36**
5.1 Introduction . . . 36
5.2 Invoking Conjoin . . . 37
5.2.1 Options . . . 38
5.3 Example . . . 39
5.4 Related Codes . . . 40

Bibliography **42**

Chapter 1

Introduction

The applications Exodiff, EPU, EJoin, and Conjoin are documented in this report. These applications are members of the Sandia Engineering Analysis Code Access System (SEACAS [5]) which is used by analysts at Sandia National Laboratories. The applications all read and/or write finite element databases stored in the EXODUS [1] format.

Each application is targeted at a few specific areas of functionality:

Exodiff compares the results data from two EXODUS files with user-specified relative or absolute tolerances. Output is either (1) a summary of the differences, or (2) a new EXODUS database where each variable is the difference of the variables in the input files. Typical uses are for verifying the results written by an application code in regression testing or for comparing the results of multiple analyses in a parameter study. Exodiff is described in Chapter 2[1].

EPU combines multiple EXODUS databases produced by a parallel application into a single EXODUS database. EPU provides several options for controlling the output database contents. Typical uses are for combining two or more processor-specific EXODUS databases into a single EXODUS database. EPU is described in Chapter 3.

EJoin joins two or more EXODUS databases into a single EXODUS database. The input databases must have disjoint meta and bulk data. That is,

- element blocks are not combined in the output model. Each element block in each input file will produce an element block in the output file. Similarly for nodesets and sidesets.
- Each node in each input file will produce a node in the output file unless one of the node matching options (`-match_node_ids` or `-match_node_coordinates`) is specified.
- Each element in each input file will produce an element in the output file.

If any of the input databases have timesteps, then the timestep values and counts must match on all databases with timesteps. EJoin is described in Chapter 4.

Conjoin joins two or more EXODUS databases into a single database. The input databases should represent the same model meta data and similar bulk data with results data written sequentially. The output database will contain the model geometry and all of the non-temporally-overlapping results data. For example, if the first database contains time data from 0 to 5

[1] Richard Drake, Sandia National Laboratories, Albuquerque, NM, was responsible for the initial design and creation of Exodiff

seconds, and the second database contains time data from 4 to 10 seconds; the output database will contain time data from 0 to 10 seconds. Typical uses are to join databases written from restarted analyses into a single database. Conjoin is described in Chapter 5.

Three of the applications above are used to join multiple EXODUS databases into a single EXODUS database, so it can be confusing to decide which code should be used.

- If the multiple EXODUS databases were created as the output from a parallel analysis code with each compute processor writing its portion of the model, then EPU would be used to join the databases.
- If the multiple EXODUS databases each contain a distinct portion of the analysis model, then EJoin would be used to join the databases.
- If the multiple EXODUS databases each contain basically the same analysis model at different times throughout the analysis event, then Conjoin would be used to join the databases.

1.1 EXODUS Concepts

The EXODUS database documentation can be accessed at http://jal.sandia.gov/SEACAS/Documentation/exodusII-new.pdf or http://sourceforge.net/projects/exodusii/files/Documentation/Documentation/exodusII-4.pdf. It contains a full detailed description of EXODUS. This section presents an overview of the concepts and structure of an EXODUS database to give some details which will be referred to in subsequent chapters. An EXODUS database contains groupings of nodes and elements. The groupings or entities include element blocks, nodesets, and sidesets.

1.1.1 Meta Data and Bulk Data

The data or information in an EXODUS database can be grouped into two major types: meta data and bulk data. The meta data is the data that gives an overview of the model and geometry that the EXODUS database contains. It is data like:

- element block, nodeset, sideset counts, ids, and names;
- the topology of the elements in an element block;
- the names of the transient variables and which entities the variables are defined on.
- shape and position of the entities, i.e., the model "topology".

The bulk data is the "problem size" data such as the ids and coordinates for each node; the ids, connectivity, and attributes for each element; the list of nodes in a nodeset and the list of elements and element faces in a sideset; the values of the transient data on each node and element.

The meta data is a higher-level view of the model and the bulk data is one realization of that model. Note that several different databases can have the same or similar meta data; but they could have vastly different bulk data. As a trivial example, the meta data for a model could be:

- a single element block named "cube" with an id of 1.
- element block "cube" contains "hex8" elements.

- element block "cube" is centered on the origin and is 10 inches per side with faces aligned with X, Y, and Z planes.
- a nodeset named "PosX" with id 1 on the positive X face.

One instance of bulk data for this meta data is an element block with 1,000,000 hexes with global ids ranging from 2 to 2,000,000 by 2. Another instance of bulk data for the same meta data could have the same element block with 8 hex elements with global ids 10,20,30,40,50,60,70,80.

Another example of multiple bulk data instances for a single meta data is decomposing an EXODUS database for a parallel run on 4 processors. Each decomposed database would have the same meta data, but the bulk data would be different on each decomposed database.

1.1.2 Database Sections

An EXODUS database is often thought of as containing multiple "sections". These include the "mesh" section which includes the bulk data describing the geometries; the element connectivities; and collections of nodes and/or (element, element-local-face) pairs which can be used for boundary conditions. Basically, all of the non-transient portions of the bulk data. Historically, this has been called the "genesis" section of the EXODUS database. The other section includes the "results" portion of the bulk data which is the global, nodal, element, nodeset, and sideset variables; and the timestep time values. The "results" portion is also called the "transient" section and is usually applied to transient data with each time step associated with a unique output state of the analysis code. The results portion is not required to be time-related data and can store information about any serialized data. For example, the calculated modal frequencies and mode shapes of the model could be stored in the "transient" section.

The EXODUS database documentation includes much more detail, but the above terminology should be helpful for some of the utility descriptions that follow in subsequent chapters.

1.2 Licensing

Each of the utilities described in this section is open-source licensed under the "New BSD License" or "Modified BSD License". The terms of the license are:

> Copyright(C) 20XX[2] Sandia Corporation. Under the terms of Contract DE-AC04-94AL85000 with Sandia Corporation, the U.S. Government retains certain rights in this software.
>
> Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
>
> - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
> - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

[2]The year is different for each of EPU, Exodiff, EJoin, and Conjoin.

- Neither the name of Sandia Corporation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The software is available at http://sourceforge.net/projects/seacas.

Chapter 2

Exodiff

2.1 Introduction

Exodiff compares the results data from two EXODUS databases. The databases should represent the same model, that is, the EXODUS meta data should be identical as should be the genesis portion of the bulk data. The only differences should be in the values of the transient bulk data. Exodiff's main purpose is to detect and report these differences. Exodiff will compare global, nodal, element, nodeset, and sideset transient variables at each selected timestep; it will also compare element attribute variables on each element block containing attributes.

If a third file is specified on the command line, it will be created with the same meta data and non-transient bulk data as the first file, and each variable in the third file will be the differences of the corresponding variables in the first two files.

A command file can be specified and used to control exactly what variables are to be compared/differenced and to what tolerance.

By default, element block names and variable names are compared ignoring case.

2.1.1 Difference Terminology

Exodiff supports several options for determining whether two values differ. These are called *difference types* and include the following:

relative difference	$\lvert val1 - val2\rvert / \max(\lvert val1\rvert, \lvert val2\rvert)$.
absolute difference	$\lvert val1 - val2\rvert$
combined difference	$\lvert val1 - val2\rvert / \max(tol, tol * \max(\lvert val1\rvert, \lvert val2\rvert))$
eigen_relative difference	$\lvert\lvert val1\rvert - \lvert val2\rvert\rvert / \max(\lvert val1\rvert, \lvert val2\rvert)$.
eigen_absolute difference	$\lvert\lvert val1\rvert - \lvert val2\rvert\rvert$
eigen_combined difference	$\lvert\lvert val1\rvert - \lvert val2\rvert\rvert / \max(tol, tol * \max(\lvert val1\rvert, \lvert val2\rvert))$

Where *tol* is a user-specified tolerance. The difference types prefixed by *eigen_* are intended to be used when the variable being differenced describes the shape of an eigenvector and the eigenvector shape is considered equal if the values on one database are equal in magnitude, but possibly of a different sign[1].

[1]Note that the difference type as implemented does not fully check whether the eigenvectors represented by the data are truly the same shape with a potential difference of sign since it works on an item-by-item basis and does

Values are considered equal if $|val1| <= floor \&\& |val2| <= floor$; where $floor$ is a user-specified value. Otherwise the difference is computed using one of the above formulas and compared to a tolerance. If the difference is greater than the tolerance, then the databases are different. At the end of execution, a summary of the differences found is output.

By default:

- All results variables and attributes are compared using a *relative difference* of 10^{-6} (about 6 significant digits) and a *floor* of 0.0.
- Nodal locations are compared using *absolute difference* with a tolerance of 10^{-6} and a *floor* of 0.0.
- Time step values are compared using *relative difference* tolerance of 10^{-6} and a floor of 10^{-15}.

2.2 Invoking Exodiff

Exodiff can be invoked using the following command lines: To do normal comparison of two files using default tolerances producing text output summarizing the differences, enter:

```
exodiff [options] [-f <cmd_file>] file1.e file2.e
```

Where ***cmd_file*** is an optional file containing options and tolerance values; its syntax is described below.

If you want Exodiff to output an EXODUS file created containing the differences of the two files, the command line is similar, but also contains the name of the file where the differences should be written[2]:

```
exodiff [options] [-f <cmd_file>] file1.e file2.e diff_file.e
```

The third invocation option reads a single file and outputs a summary of the variable data contained in the file. The summary data are the minimum and maximum values for each variable and the time step and entity id where the minimums and maximums occurred. This file can be used for preparing a command input file for use in the previous two invocations. The `no_coord_sep` option if present will inhibit the calculation and output of the minimum distance between any two nodes in the file which can take a long time for large models and is often unneeded data.

```
exodiff -summary [no_coord_sep] file.e (create variable summary)
```

The remaining invocation lines will output a short usage summary, a much longer usage summary, and the last just output the version information.

```
exodiff [-h] [-help]            (short usage summary)
exodiff [-H]                    (longer usage summary)
exodiff [-v] [-version]         (version info)
```

not check whether all items in the first database are multiplied by the same 1.0 or -1.0 to match the items in the second database. However, the implementation can be improved in the future without breaking any existing scripts or command files.

[2]Note that all variables on the third file are the difference of the values on the first two files including the displacement variables. If you visualize the file containing the differences, the visualization program may show a strange deformed shape since the displacement variables are no longer true displacements.

The basic behavior can be modified using several optional parameters specified on the command line. These are documented below:

2.2.1 Optional Parameters

-t `<real value>`	Overrides the default tolerance of 10^{-6} for all variables.
-F `<real value>`	Overrides the default floor tolerance of 0.0 for all variables.
-absolute	Use absolute differences as default tolerance type
-relative	Use relative differences as default tolerance type
-combined	Use combined differences as default tolerance type
-eigen_absolute	Use eigen_absolute differences as default tolerance type (absolute value of values)
-eigen_relative	Use eigen_relative differences as default tolerance type (absolute value of values)
-eigen_combined	Use eigen_combined differences as default tolerance type (absolute value of values)
-T `<offset>`	Match timestep 'x+offset' in first file with timestep 'x' in second file.
-TA	Automatically determine the timestep offset such that both databases end at the same step.
-TM	Automatically determine the timestep offset to find the closest match in file1 to the first step on file2.
-q	Quiet. Only errors will be sent to stdout. Comparison mode will echo: `exodiff: Files are the same.` or `exodiff: Files are different.`
-show_all_diffs	Show all differences for all variables, not just the maximum. Default behavior is that there will be a maximum of one difference output per variable per timestep. If this option is specified, then any pair of values that exceed the tolerance will be output. Use of this option Can result in lots of output on large files with lots of differences.
-m	Invoke a matching algorithm to create a mapping between the nodes and elements of the two files based on geometric proximity. The topology must still be the same (within tolerance), but can be ordered differently. A match must be found for all nodes and elements or Exodiff will output an error message and stop.
-p	Invoke a matching algorithm similar to the -m option. However this option ignores unmatched nodes and elements. This allows comparison of files that only partially overlap.
-match_ids	Invoke a matching algorithm which matches nodes and elements using the node and element global id maps in the two files. This is the default mode of operation.

-match_file_order	Invoke a matching algorithm using the node and element position order in the two files.
-show_unmatched	If the -p option is given, this prints out the elements that did not match.
-dumpmap	If the -m or -p switch is given, this prints out the resulting map between the nodes and elements in the two files.
-nsmap	Create a map between the nodeset nodes in the two files if they include the same nodes, but are in different order. This is enabled by default.
-ssmap	Create a map between the sideset faces in the two files if they include the same sides, but are in different order. This is enabled by default.
-no_nsmap	Compare nodeset nodes based on file order only.
-no_ssmap	Compare sideset faces based on file order only.
-s	Short block type compare. Forces element block type strings to be compared only up to the shortest string length. For example, "HEX" and "HEX8" will be considered the same. This is enabled by default.
-no_short	Do not do the short block type compare. Forces element block type strings to fully match. For example, "HEX" and "HEX8" will be considered different.
-ignore_case	Ignore case. Variable names are compared case in-sensitive. For example, "Stress" and "STRESS" will be considered as the same variable. This is enabled by default.
-case_sensitive	Variable names are compared case sensitive. For example, "Stress" and "STRESS" will be considered as two different variables.
-ignore_maps	Output node and element difference summaries using file local implicit ids instead of global ids. Note that the matching of nodes and elements will use the mapping option specified above; this option only affects the output of the node or element id where the difference occurred.
-ignore_nans	Don't check data for NaNs. By default, Exodiff will output a warning message if any variable's value is NaN (Not a number).
-ignore_dups	If two elements/nodes are in the same location in match or partial match case, return first match instead of aborting. This is used in the -m and -p matching options. Normally, Exodiff will output an error message if a node in one file can be matched to two or more nodes in the second file.
-ignore_attributes	Don't compare element attribute values.

-nosymm	Turn off symmetric variable name checking. By default, a warning will be produced if a variable that is not to be excluded[3] is contained in the second file given on the command line but not the first. This "symmetric" check can be turned off with this option and the extra variables in the second file will be ignored.
-allow_name_mismatch	Allow a variable name that is in the first database to be absent from the second database. The default behavior is to output an error if all variables in the first file cannot be matched to variables in the second file.
-x `<list>`	Exclude time steps. Does not calculate any variable differences for the time steps given in the list of integers. The format is comma separated and ranged integers (with no spaces), such as "1,5-9,28". The first time step is the number one.
-steps `<b:e:i>`	Specify subset of steps to consider. Syntax is begin:end:increment, Enter -1:: for just the last step. If only begin is set, then end=begin and only that step will be considered
-norms	Calculate the L_2 norm of variable differences and output if the norm is greater than 0.0. The output will also contain the L_2 norm of each variable. This can be used to give an idea of the relative magnitudes of the differences compared to the magnitudes of the variables. This is for informational purposes only at this time; it does not affect the determination of whether the databases compare the same or different.
-stat	Return an exit status of 2 if the files are different. Normally, the exit status is zero unless an error occurs.
-maxnames `<int>`	There is a compiled limit of 1000 exodus variable names. This option allows the maximum number to be set to a larger value if either of the input databases contains more than 1000 variables.
-use_old_floor	When Exodiff was first released, it used an incorrect definition of the floor tolerance in which the differences were ignored if the difference itself was less than the floor value. This was fixed several years ago to the new definition which is to ignore the differences if $\|a\| < floor \&\& \|b\| < floor$. This option was provided so that users could use the old definition if desired. It should not be used.
-summary	Produce a summary in Exodiff input format. This will create an output file with max/min statistics on the data in the format of an Exodiff input file. The algorithm to determine the minimum separation between any two nodes can be disabled with the "no_coord_sep" switch.
-copyright	Output the copyright and license information.
-f `<cmd file>`	Use the given file to specify the variables to be considered and the tolerances to be used for each variable type or each individual variable. See Section 2.2.2 for details of the syntax in this file.

[3] See the command file description in Section 2.2.2 for details on excluding variables

-H file	Show the syntax help for the command input file. This is also documented in Section 2.2.2.

2.2.2 Exodiff Command File Syntax

If an Exodiff invocation uses the `-f <cmd_file>` option, then Exodiff will read commands from the specified file in addition to parsing the options given on the command line. The command line will be parsed first and then the commands in the input file. The primary use of the input file is to give more control over the difference types and tolerances to be used for individual variables.

The basic syntax of the file is:

- each command is given on a separate line.
- Anything following the `#` character on a line will be treated as a comment and ignored.
- Within a "variables" block, lines must be indented and must begin with a "tab" character.

The valid command lines are shown in all uppercase in the following list. The list also describes the behavior that the command line will specify.

- The variable names are case insensitive (unless the `-case_sensitive` option is specified or there is a `CASE SENSITIVE` line in the command file).
- All keyword comparisons are case insensitive. Abbreviations can be used.
- All variable comparisons use the default of relative 10^{-6} for variables and absolute 10^{-6} for coordinates. This is overridden with the `DEFAULT TOLERANCE` line. The `DEFAULT TOLERANCE` values are overridden by the values given on the `VARIABLES` line and apply only to those variables. Each variable can override all values by following its name with a value.
- A variable name must start with a tab character. If there is at least one variable name of a specified type (element, nodal, global, ...) is listed, then only the listed variable(s) of that type will be differenced. The variable name can be followed by an optional difference type and tolerance, and an optional `floor` and floor tolerance. The NOT symbol `!` means do not include this variable. Mixing non-! and ! is not allowed without the `(all)` specifier. For example

```
NODAL VARIABLES (all) absolute 1.E-8
<tab> DISPLX
<tab> !VELX
<tab> VELY relative 1.E-6 floor 1.e-10
```

  In this case, all variables are considered that are not prepended with a "!" symbol.
- If a variable type (e.g. `NODAL VARIABLES`) is not specified, no variables of that type will be considered. Allowed variable types are: `GLOBAL VARIABLES`, `NODAL VARIABLES`, `ELEMENT VARIABLES`, `NODESET VARIABLES`, and `SIDESET VARIABLES`.
- The command line option to set the maximum number of Exodus names can be set with `MAX NAMES <int>`. Note: this option must appear before the variable blocks are read!
- The time step exclusion option can be used in the input file with the syntax `EXCLUDE TIMES <list>`, where `<list>` has the same format as in the command line options.

- The matching algorithm, `-m`, can be turned on from the input file with the `APPLY MATCHING` keyword on a separate line.
- The nodeset matching algorithm, `-nsmap`, can be turned on from the input file with the `NODESET MATCH` keyword on a separate line.
- The sideset matching algorithm, `-ssmap`, can be turned on from the input file with the `SIDESET MATCH` keyword on a separate line.
- The short block type compare option, `-s`, can be turned on with the `SHORT BLOCKS` keyword.
- The no short compare option, `-no_short`, can be turned on with the `NO SHORT BLOCKS` keyword.
- The case_sensitive option, `-case_sensitive`, can be turned on with the `CASE SENSITIVE` keyword.
- The ignore case option, `-i`, can be turned on with the `IGNORE CASE` keyword. (default behavior)
- The ignore maps option, `-ignore_maps`, can be turned on with the `IGNORE MAPS` keyword.
- The ignore nans option, `-ignore_nans`, can be turned on with the `IGNORE NANS` keyword.
- The ignore dups option, `-ignore_dups`, can be turned on with the `IGNORE DUPLICATES` keyword.
- The time step offset option, `-T`, can be turned on with the `STEP OFFSET` keyword.
- The automatic time step offset option, `-TA`, can be turned on with the `STEP OFFSET AUTOMATIC` keyword.
- The automatic time step offset option, `-TM`, can be turned on with the `STEP OFFSET MATCH` keyword.
- The calculation of the L2 norm of differences `-norms`, can be turned on with the `CALCULATE NORMS` keyword.
- The exit status return option, `-stat`, can be turned on with the `RETURN STATUS` keyword.

2.3 Examples

The output below shows an example run of Exodiff. The command invocation used was:

```
exodiff -f P_exodiff.cmd P_gold_results.e bar-P.e
```

The ***P_exodiff.cmd*** command file contains the following:

```
COORDINATES absolute 1.e-6

TIME STEPS relative 1.e-6 floor 0.0

GLOBAL VARIABLES relative 1.e-6 floor 1.e-16
internal_energy
```

```
kinetic_energy
momentum_x

NODAL VARIABLES relative 1.e-4 floor 1.e-16
displacement_x
acceleration_x
force_internal_x
mass
velocity_x

ELEMENT VARIABLES relative 1.e-6 floor 1.e-16
eqps
stress_xx          absolute 1000
stress_yy          absolute 1000
stress_zz          absolute 1000
temperature        absolute 1
yield_stress       absolute 1000
```

The first section of the output shows the code version and contact information and when the output was generated; followed by some summary statistics of the two files including the file paths and the counts of nodes, elements, etc. If options are read from a command file, the path to that file is listed.

```
   ****************************************************************
     EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF

                       Version 2.43 (2011-04-07)
              Authors:  Richard Drake, rrdrake@sandia.gov
                        Greg Sjaardema, gdsjaar@sandia.gov
                        2011/04/28   21:11:00 MDT

     EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF
   ****************************************************************

Reading first file ...
Reading second file ...
  FILE 1: /home/exodiff/axial_pulse_par_ns/P_gold_results.e
   Title: Default Database Title
          Dim = 3, Blocks = 1, Nodes = 816, Elements = 450, Nodesets = 6, Sidesets = 0
          Vars: Global = 7, Nodal = 13, Element = 16, Nodeset = 0, Sideset = 0, Times = 23

  FILE 2: /home/exodiff/axial_pulse_par_ns/bar-P.e
   Title: Default Database Title
          Dim = 3, Blocks = 1, Nodes = 816, Elements = 450, Nodesets = 6, Sidesets = 0
          Vars: Global = 7, Nodal = 13, Element = 16, Nodeset = 0, Sideset = 0, Times = 23

  COMMAND FILE: /home/exodiff/axial_pulse_par_ns/P_exodiff.cmd
```

The next output section summarizes what variables will be compared and the difference types, tolerances, and floor values that will be used. Note that the command file is specifying that only a

subset of the variables on the files will be differenced since the output above shows 7 global variables, 13 nodal variables, and 16 element variables, but the list below only shows 3 global, 5 nodal, and 6 element variables.

```
Coordinates will be compared .. tol:    1e-06 (absolute), floor:        0
Time step values will be compared .. tol:    1e-06 (relative), floor:        0
Global variables to be compared:
internal_energy                  tol:    1e-06 (relative), floor:    1e-16
kinetic_energy                           1e-06 (relative),           1e-16
momentum_x                               1e-06 (relative),           1e-16
Nodal variables to be compared:
displacement_x                   tol:    1e-06 (relative), floor:    1e-16
acceleration_x                           1e-06 (relative),           1e-16
force_internal_x                         1e-06 (relative),           1e-16
mass                                     1e-06 (relative),           1e-16
velocity_x                               1e-06 (relative),           1e-16
Element variables to be compared:
eqps                             tol:    1e-06 (relative), floor:    1e-16
stress_xx                                1e-06 (relative),           1e-16
stress_yy                                1e-06 (relative),           1e-16
stress_zz                                1e-06 (relative),           1e-16
temperature                              1e-06 (relative),           1e-16
yield_stress                             1e-06 (relative),           1e-16
No Element Attribute variables on either file.
No Nodeset variables on either file.
No Sideset variables on either file.

  ================================================================
  NOTE: All node and element ids are reported as global ids.
```

The next output section shows the results of the differencing. For the first several timesteps, no differences were found.

```
  --------- Time step 1, 0.0000000e+00 ~ 0.0000000e+00, rel diff:  0.00000e+00 ---------
Global variables:
Nodal variables:
Element variables:
  --------- Time step 2, 2.2708229e-08 ~ 2.2708229e-08, rel diff:  0.00000e+00 ---------
Global variables:
Nodal variables:
Element variables:
  --------- Time step 3, 8.1607527e-08 ~ 8.1607527e-08, rel diff:  0.00000e+00 ---------
Global variables:
Nodal variables:
Element variables:
  --------- Time step 4, 2.3437714e-07 ~ 2.3437714e-07, rel diff:  0.00000e+00 ---------
Global variables:
Nodal variables:
Element variables:
```

```
... deleted some output ...

  --------- Time step 11, 7.7253933e-06 ~ 7.7253933e-06, rel diff:  0.00000e+00 ---------
Global variables:
Nodal variables:
Element variables:
  --------- Time step 12, 8.9485520e-06 ~ 8.9485520e-06, rel diff:  0.00000e+00 ---------
Global variables:
Nodal variables:
Element variables:
```

At this time step, differences are detected and output. The output format is:

```
  variable name  diff type   val file 1       val file 2     difference  (which entity)
   stress_xx     rel diff: -1.1444528e+04 ~ -1.1444553e+04 = 2.15241e-06 (block 1, elmt 66)
```

Note that only the maximum difference found for each variable at each time step is output. There may be many more differences detected.

```
  --------- Time step 13, 1.0171704e-05 ~ 1.0171704e-05, rel diff:  1.66547e-16 ---------
Global variables:
Nodal variables:
   acceleration_x    rel diff:  1.1719403e+04 ~  1.1719426e+04 = 1.98010e-06 (node 68)
   force_internal_x  rel diff: -5.8141261e+02 ~ -5.8141376e+02 = 1.98010e-06 (node 68)
Element variables:
   stress_xx     rel diff: -1.1444528e+04 ~ -1.1444553e+04 = 2.15241e-06 (block 1, elmt 66)
   stress_yy     rel diff: -4.9048081e+03 ~ -4.9048309e+03 = 4.63816e-06 (block 1, elmt 266)
   stress_zz     rel diff: -4.9048129e+03 ~ -4.9048357e+03 = 4.64075e-06 (block 1, elmt 266)
  --------- Time step 14, 1.1394849e-05 ~ 1.1394849e-05, rel diff:  1.48669e-16 ---------
Global variables:
Nodal variables:
   displacement_x    rel diff:  1.0981488e-11 ~  1.0980936e-11 = 5.02741e-05 (node 740)
   acceleration_x    rel diff:  2.0947516e+02 ~  2.0950905e+02 = 1.61776e-04 (node 639)
   force_internal_x  rel diff: -5.1961477e+00 ~ -5.1969885e+00 = 1.61776e-04 (node 639)
   velocity_x        rel diff:  5.9451636e-05 ~  5.9447122e-05 = 7.59215e-05 (node 740)
Element variables:
   stress_xx     rel diff: -1.9233572e+02 ~ -1.9236707e+02 = 1.62922e-04 (block 1, elmt 326)
   stress_yy     rel diff: -8.2409892e+01 ~ -8.2442564e+01 = 3.96299e-04 (block 1, elmt 326)
   stress_zz     rel diff: -8.2408238e+01 ~ -8.2440733e+01 = 3.94165e-04 (block 1, elmt 326)
  --------- Time step 15, 1.2617989e-05 ~ 1.2617989e-05, rel diff:  1.34258e-16 ---------
Global variables:
Nodal variables:
   displacement_x    rel diff:  1.4026865e-13 ~  1.4075811e-13 = 3.47725e-03 (node 648)
   acceleration_x    rel diff:  2.7998765e+00 ~  2.8278524e+00 = 9.89300e-03 (node 700)
   force_internal_x  rel diff: -1.3890498e-01 ~ -1.4029290e-01 = 9.89300e-03 (node 700)
   velocity_x        rel diff:  7.8395852e-07 ~  7.8796013e-07 = 5.07844e-03 (node 648)
Element variables:
   stress_xx     rel diff: -2.4472564e+00 ~ -2.4778807e+00 = 1.23591e-02 (block 1, elmt 386)
```

```
   stress_yy     rel diff: -1.0366468e+00 ~ -1.0631157e+00 = 2.48974e-02 (block 1, elmt 386)
   stress_zz     rel diff: -1.0455424e+00 ~ -1.0720140e+00 = 2.46934e-02 (block 1, elmt 386)

... deleted some output ...

  --------- Time step 22, 2.1179859e-05 ~ 2.1179859e-05, rel diff:  7.99848e-16 ---------
Global variables:
Nodal variables:
Element variables:
  --------- Time step 23, 2.2036041e-05 ~ 2.2036041e-05, rel diff:  7.68771e-16 ---------
Global variables:
Nodal variables:
Element variables:
```

The final section is the status output indicating that differences were detected. This string will not change in future versions and can be searched for to determine whether the files are the same or different. The Exodiff exit status can also be used for this if the `-status` option is set.

```
exodiff: Files are different
```

The next example shows the summary output produced by the command line:

```
exodiff -summary bar-P.e
```

```
#  ****************************************************************
#    EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF
#
#                       Version 2.43 (2011-04-07)
#             Authors:  Richard Drake, rrdrake@sandia.gov
#                       Greg Sjaardema, gdsjaar@sandia.gov
#                       2011/06/03   11:23:07 MDT
#
#    EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF  EXODIFF
#  ****************************************************************

#  FILE 1: /scratch/user/bar-P.e
#   Title: An Exodiff Summary Example
#          Dim = 3, Blocks = 1, Nodes = 204, Elements = 50, Nodesets = 5, Sidesets = 0
#          Vars: Global = 7, Nodal = 10, Element = 16, Nodeset = 0, Sideset = 0, Times = 206


# ============================================================
#  NOTE: All node and element ids are reported as global ids.

# NOTES:  - The min/max values are reporting the min/max in absolute value.
#         - Time values (t) are 1-offset time step numbers.
#         - Element block numbers are the block ids.
#         - Node(n) and element(e) numbers are 1-offset.
```

```
COORDINATES absolute 1.e-6    # min separation = 0.1

TIME STEPS relative 1.e-6 floor 0.0     # min:               0 @ t1 max:   2.2088109e-05 @ t206

GLOBAL VARIABLES relative 1.e-6 floor 0.0
external_energy  # min:               0 @ t1 max:               0 @ t1
internal_energy  # min:               0 @ t1 max:        22205882 @ t206
kinetic_energy   # min:               0 @ t1 max:        20210551 @ t206
momentum_x       # min:               0 @ t1 max:       42651.567 @ t206
momentum_y       # min:               0 @ t1 max:               0 @ t1
momentum_z       # min:               0 @ t1 max:               0 @ t1
timestep         # min:               0 @ t1 max:   1.3153439e-07 @ t51

NODAL VARIABLES relative 1.e-6 floor 0.0
acceleration_x    # min:               0 @ t1,n1 max:   3.7989521e+08 @ t190,n1
acceleration_y    # min:               0 @ t1,n1 max:               0 @ t1,n1
acceleration_z    # min:               0 @ t1,n1 max:               0 @ t1,n1
force_internal_x  # min:               0 @ t1,n1 max:        82739542 @ t190,n5
force_internal_y  # min:               0 @ t1,n1 max:   1.3526743e+08 @ t206,n201
force_internal_z  # min:               0 @ t1,n1 max:   1.3526743e+08 @ t206,n201
mass              # min:        0.111625 @ t1,n1 max:         0.22325 @ t1,n21
velocity_x        # min:               0 @ t1,n1 max:       994.29394 @ t206,n201
velocity_y        # min:               0 @ t1,n1 max:               0 @ t1,n1
velocity_z        # min:               0 @ t1,n1 max:               0 @ t1,n1

ELEMENT VARIABLES relative 1.e-6 floor 0.0
eqps                   # min:               0 @ t1,b2,e1 max:   0.00083689614 @ t206,b2,e50
rate_of_deformation_xx  # min:               0 @ t1,b2,e1 max:       539.94599 @ t116,b2,e50
rate_of_deformation_yy  # min:               0 @ t1,b2,e1 max:   4.0126165e-32 @ t200,b2,e46
rate_of_deformation_zz  # min:               0 @ t1,b2,e1 max:   1.1275579e-32 @ t185,b2,e49
rate_of_deformation_xy  # min:               0 @ t1,b2,e1 max:   1.1868359e-13 @ t186,b2,e42
rate_of_deformation_yz  # min:               0 @ t1,b2,e1 max:   1.1093839e-32 @ t185,b2,e49
rate_of_deformation_zx  # min:               0 @ t1,b2,e1 max:   4.7474497e-14 @ t203,b2,e39
sound_speed            # min:          394000 @ t1,b2,e1 max:       395751.09 @ t206,b2,e50
stress_xx              # min:               0 @ t1,b2,e1 max:    3.980196e+09 @ t206,b2,e50
stress_yy              # min:               0 @ t1,b2,e1 max:   2.7099141e+09 @ t206,b2,e50
stress_zz              # min:               0 @ t1,b2,e1 max:   2.7099141e+09 @ t206,b2,e50
stress_xy              # min:               0 @ t1,b2,e1 max:   2.7908475e-07 @ t205,b2,e46
stress_yz              # min:               0 @ t1,b2,e1 max:   3.7940314e-26 @ t190,b2,e49
stress_zx              # min:               0 @ t1,b2,e1 max:   1.5648498e-07 @ t188,b2,e49
temperature            # min:             298 @ t78,b2,e8 max:       299.37791 @ t206,b2,e50
yield_stress           # min:   7.5751705e+08 @ t142,b2,e34 max:   1.3389387e+09 @ t148,b2,e50

# No NODESET VARIABLES

# No SIDESET VARIABLES
```

The output starts with a database summary similar to the previous example. It then gives a summary of the minimum and maximum values of each variable and the timestep and node or element where

that minimum or maximum occurs.

The format of the summary is such that it can be used as a basis for creating an Exodiff command input file.

Chapter 3

EPU

3.1 Introduction

One of the typical processes for performing parallel analyses with EXODUS databases is to decompose the finite element model into multiple pieces such that each processor can read and write its own portion of the finite element model and results data. For example, if a parallel analysis is to be run on the mesh file ***mesh.g*** using 8 processors, then ***mesh.g*** will be decomposed into 8 pieces or submeshes: ***mesh.g.8.0***, ***mesh.g.8.1***, ..., ***mesh.g.8.7***. Each submesh will contain a subset of the nodes and elements of the entire mesh and some communication data indicating which nodes and elements are on the boundary of this submesh and the submesh of one or more other processors.

The analysis code is then executed in parallel and each processor reads its portion of the mesh from its respective submesh; when it outputs results and/or restart data, it creates a new file containing its portion of the submesh and the results that are calculated on that submesh. An "N" processor run will create "N" separate files for each results and/or restart "dataset" that it creates.

The analyst may want to visualize or postprocess the data in the submeshes as a single mesh, so each submesh needs to be joined together to create a single "global" file containing all of the data.[1]

This joining together of parallel submeshes is the purpose of EPU. It will read the data from each submesh and map it into the correct location in the "global" file; discarding duplicate data as required.

The name EPU is an acronym for EXODUS *Parallel Unification* Program. The "alternate" meaning of EPU is *E Pluribus Unum* which means "Out of Many One" which is described in more detail at http://www.greatseal.com/mottoes/unum.html.

3.2 Invoking EPU

EPU is typically invoked using a command line similar to:

```
epu [options] basename
epu [options] -auto full_filename
```

[1] Note that some visualizers can handle the separate, per-processor files, so joining is not required in all cases.

Where `basename` is the "base" portion of the filename. For example, if one of the files to be joined is ***results.e.8.3***, then the basename would be specified as ***results***.

3.2.0.1 File Naming Convention

The naming of the per-processor files must follow the convention that the last two dot-separated suffixes of the filename represent the total processor count and the zero-based processor rank of the processor that created the file, respectively. Also, the last suffix (rank) must be zero-filled such that the width of the field is the same as the width required to represent the processor count. For example, if the files were written for a 1000 processor run, the files would be ***basename.e.1000.0000*** through ***basename.e.1000.0999***.

3.2.0.2 Defaults and Requirements

The default behavior is that EPU will join all per-processor files into the output file; all transient variables (global, nodal, element, nodeset, and sideset) will be transferred for all timesteps. It is assumed that all per-processor files represent a portion of the same overall finite element model and that they all have the same EXODUS meta data including variable names. In addition, each file must have the same timesteps.

Several options are available to modify the default execution of EPU. These are documented below.

3.2.1 Options

3.2.1.1 Basic invocation options

The majority of the time, the filename-related options to EPU can be determined automatically by passing the `-auto` option and instead of the basename, the full filename of one of the files is specified. For example, given files of the form: ***/scratch/user/results.e.16.00***, ***/scratch/user/results.e.16.01***, ..., ***/scratch/user/results.e.16.15***; EPU could be invoked as:

```
epu -auto /scratch/user/results.e.16.00
```

and the output file would be written to ***results.e*** in the current directory.

3.2.1.2 Disk-related filename options

If this simple invocation does not work for some reason, then the following options can be used for full control over the reading and writing of the files. EPU will read and write files of the form:

```
Reads:  root#o/sub/basename.suf.#p.0 to
        root(#o+#p)%#r/sub/basename.isuf.#p.#p-1
Writes: current_directory/basename.osuf
```

Where:

current_directory By default, this is the directory from which EPU is being run. It can be set via the `-current_directory <val>` option.

basename The base portion of the filename before the suffices and after the root/sub portion (if any).

isuf The suffix of the input files not including the processor-specific data. For example, given the file ***results.e.8.0***, the suffix would be `e`.

osuf The suffix that should be used for the output file.

#p The number of processors the files were decomposed for.

root The "non-raid" portion of the complete pathname (see below). If not on a raid system, this can be omitted.

sub The portion of the pathname following the raid data (see below). If not on a raid system, this can be omitted.

#o The "raid offset"

#r The "raid count" which is the number of raid disks.

The raid options are not used very often, but can be very useful if your compute system uses a filesystem that requires it. On some compute systems, a parallel job needs to spread its data across multiple filesystems in order to load balance the IO requests. The filesystem mount points will typically look something like: /scratch0, /scratch1, /scratch2, /scratch3. Within these filesystems, the directory hierarchy will typically match. So, for example, assume that a 16 processor job is run on a filesystem like this and the files to be joined are named:

/scratch0/user/results.e.16.00	/scratch1/user/results.e.16.01
/scratch2/user/results.e.16.02	/scratch3/user/results.e.16.03
/scratch0/user/results.e.16.04	/scratch1/user/results.e.16.05
/scratch2/user/results.e.16.06	/scratch3/user/results.e.16.07
/scratch0/user/results.e.16.08	/scratch1/user/results.e.16.09
/scratch2/user/results.e.16.10	/scratch3/user/results.e.16.11
/scratch0/user/results.e.16.12	/scratch1/user/results.e.16.13
/scratch2/user/results.e.16.14	/scratch3/user/results.e.16.15

If the user is in some other directory and wants the output file written to /scratch0/user/results.epu, the command to do this would be:

```
epu -extension e -output_extension epu -raid_count 4
-processor_count 16 -Root_directory /scratch
-Subdirectory user -current_directory
/scratch0/user  results
```

If for some reason, the processor 0 file had been written to the ***/scratch2*** filesystem and then subsequent files followed the cycle, then the additional option `-offset 2` specifying the "raid offset" would be added.

The options that control this are:

`-auto`	Automatically set Root, Proc, Ext from the name of one of the files being joined. The filename is used in place of "basename".
`-extension <val>`	EXODUS database extension for the input files
`-output_extension <val>`	EXODUS database extension for the output file

`-offset <val>`	Raid Offset
`-raid_count <val>`	Number of raids
`-processor_count <val>`	Number of processors
`-current_directory <val>`	Current directory
`-Root_directory <val>`	Root directory
`-Subdirectory <val>`	Subdirectory containing input exodusII files

Note that the majority of the time, this complicated option setting is not required and everything can be set automatically via the `-auto` option.

3.2.1.3 Additional Options

`-help`	Print the version and a summary of all options and exit.
`-version`	Print version and exit.
`-map`	Map element ids to original order if possible [default].
`-nomap`	Do not map element ids to original order.
`-debug <val>`	Debug level (values are added together). 1 = Timing information. 2 = Check consistent nodal field values between processors. 4 = Verbose Element block information. 8 = Check consistent nodal coordinates between processors. 16 = Verbose Sideset information. 32 = Verbose Nodeset information. 64 = Put exodus library into verbose mode. 128 = Check consistent global field values between processors.
`-width <val>`	Width of output screen, default = 80.
`-add_processor_id`	Add 'processor_id' element variable to the output file. The value of the variable is the processor on which that element existed.
`-append`	Append to an existing database instead of overwriting an existing database. Timestep transfer will start after last timestep on the existing database.
`-steps <val>`	Specify a subset of timesteps to transfer to output file. Format is begin:end:step. For example, -steps 1:10:2 would result in steps 1,3,5,7,9 being transferred to the output databaes. Enter LAST to just transfer last step, for example, "-steps LAST"

`-Part_count <val>` How many pieces (files) of the model should be joined. This option is typically used with either the `-start_part` or the `-subcycle` options in the case where the user wants to only join a subset of the individual per-processor files.

`-start_part <val>` Start with processor n file (0-based). Used with the `-Part_count` option

`-subcycle [val]` Subcycle. Create 'val' subparts if 'val' is specified. Otherwise, create multiple parts each of size 'Part_count'. The subparts can then be joined by a subsequent run of EPU. This option is usually used for one of two cases:

1. The maximum number of simultaneously open files on the filesystem or system is less than the processor count[2]. In that case, unless this option is used, EPU will have to open and close each file for each time it is accessed which can take a long time.

2. The user wants to use a parallel visualization program that can handle a model split into multiple parts, but wants fewer parts than were used in the parallel analysis run.

If the parameters result in the creation of "n" parts, then the output files written will be ***basename.osuf.n.0*** through ***basename.osuf.n.n-1***.

`-gvar <val>` Comma-separated list of global variables to be joined or ALL or NONE. The default behavior is that all global variables will be written to the output file.

`-evar <val>` Comma-separated list of element variables to be joined or ALL or NONE. Variables can be limited to certain blocks by appending a colon followed by the block id. For example, `-evar sigxx:10:20` would result in the variable `sigxx` would be written for element blocks with id 10 and 20. The default behavior is that all element variables will be written to the output file.

`-nvar <val>` Comma-separated list of nodal variables to be joined or ALL or NONE. The default behavior is that all nodal variables will be written to the output file.

`-nsetvar <val>` Comma-separated list of nodeset variables to be joined or ALL or NONE. The default behavior is that all nodeset variables will be written to the output file.

`-ssetvar <val>` Comma-separated list of sideset variables to be joined or ALL or NONE. The default behavior is that all sideset variables will be written to the output file.

`-omit_nodesets` Don't transfer nodesets to output file.

`-omit_sidesets` Don't transfer sidesets to output file.

`-copyright` Show copyright and license data.

[2] EPU will output a message to standard output if this happens.

When EPU is invoked, it will parse the user-specified options from the command line and it will also see if the environment variable EPU_OPTIONS exists. If the variable exists, then it will be parsed first followed by the parsing of the options on the command line.

3.3 Example

The output below shows the results of running the following command:

```
epu -auto epu_example.rsout.8.0
```

The first section of the output shows the code version followed by some information showing how the -auto option decoded the filename and automatically set some of the filename-related options. It then shows which files will be used in the joining process.

```
epu --  E Pluribus Unum
(Out of Many One -- see http://www.greatseal.com/mottoes/unum.html)
ExodusII Parallel Unification Program
(Version: 3.30) Modified: 2011/01/12

The following options were determined automatically:
 basename = 'epu_example'
-processor_count 8
-extension rsout
-Root_directory

Input(0): './epu_example.rsout.8.0'
...
Input(7): './epu_example.rsout.8.7'
```

The next section shows the progress of reading the meta data information from each file and creating the output file and populating the mesh portion of the bulk data.

```
**** READ LOCAL (GLOBAL) INFO ****
Node map is contiguous.
Finished reading/writing Global Info


**** GET BLOCK INFORMATION (INCL. ELEMENT ATTRIBUTES) ****
Global block count = 3

Getting element block info.
Element id map is contiguous.

**** GET SIDE SETS *****

**** GET NODE SETS *****
```

```
**** BEGIN WRITING OUTPUT FILE *****
Output:   './epu_example.rsout'
Writing global node number map...
Writing out master global elements information...

Reading and Writing element connectivity & attributes



**** GET COORDINATE INFO ****
Wrote coordinate names...
Wrote coordinate information...
```

At this point, the non-transient portion of the output file is complete and all that remains is defining the results data and transferring it to the output file. The timing information in the timestep transfer gives an estimate of how long the transfer is expected to take; its accuracy can vary depending on the machine load and the file system load.

At the end, a summary of the output mesh is given.

```
**** GET VARIABLE INFORMATION AND NAMES ****
Found 11 global variables.
cumulative_topology_modification  current_eigenvalue
delta_t                           hourglass_energy
last_rebalance_comm_load          last_rebalance_step
stepcount                         strain_external_energy
timestep_lanczos_last             timestep_nodal
updated_step_count

Found 14 nodal variables.
displacement_x                    displacement_y
displacement_z                    3by3_block_diag_pc_xx
acceleration_x                    acceleration_y
acceleration_z                    force_constraint_x
force_constraint_y                force_constraint_z
temperature                       velocity_x
velocity_y                        velocity_z

Found 11 element variables.
dilmod                        element_mass
stress_xx                     stress_yy
stress_zz                     stress_xy
stress_yz                     stress_zx
temperature                   timestep
volume


**** GET TRANSIENT NODAL, GLOBAL, AND ELEMENT DATA VALUES ****
```

```
Number of time steps on input databases = 93

Transferring step 1 to step 93 by 1
Wrote step  1, time 1.0000e-05 [  1.1%, Elapsed=1.1s, ETA=1.7m]
Wrote step  2, time 1.0000e-02 [  2.2%, Elapsed=2.2s, ETA=1.7m]
Wrote step  3, time 2.0000e-02 [  3.2%, Elapsed=3.3s, ETA=1.7m]

... deleted ...

Wrote step 90, time 8.9000e-01 [ 96.8%, Elapsed=3.5m, ETA=7.1s]
Wrote step 91, time 9.0000e-01 [ 97.8%, Elapsed=3.6m, ETA=4.7s]
Wrote step 92, time 9.1000e-01 [ 98.9%, Elapsed=3.6m, ETA=2.4s]
Wrote step 93, time 9.2000e-01 [100.0%, Elapsed=3.7m, ETA=0.0s]
******* END *******
IO Word size is 8 bytes.
 Title: EPU Example File

 Number of coordinates per node       =        3
 Number of nodes                      =    35946
 Number of elements                   =    28913
 Number of element blocks             =        3

 Number of nodal point sets           =        3
 Number of element side sets          =        3
```

3.4 Related Codes

EPU replaces the functionality of the `nem_join` [4] code. EPU provides a superset of the `nem_join` functionality and is also much faster in almost all cases. EPU also supports nodeset and sideset variables and the naming of element blocks, nodesets, and sidesets which are not supported by `nem_join`.

Chapter 4

EJoin

4.1 Introduction

EJoin is used to join two or more EXODUS databases into a single EXODUS database. The input databases must have disjoint meta and bulk data. That is,

- element blocks are not combined in the output model. Each element block in each input file will produce an element block in the output file. Similarly for nodesets and sidesets.
- Each node in each input file will produce a node in the output file unless one of the node matching options (`-match_node_ids` or `-match_node_coordinates`) is specified.
- Each element in each input file will produce an element in the output file. Elements are never combined even if all of the nodes on two elements are combined, the output file would have two elements with identical connectivity which is usually not desired.

If any of the input databases have timesteps, then the timestep values and counts must match on all databases with timesteps.

4.2 Invoking EJoin

The minimal command line for executing EJoin is:

```
ejoin {list of files to join}
```

This would create an output mesh named ***ejoin-out.e*** containing the entities and data from the files listed on the command line.

Several options are available to modify the default behavior of EJoin. In the description of the options, there is the concept of a "part" which is simply the mesh from an input file. The parts are numbered sequentially starting at 1.

4.2.1 Options

`-help`	Print this summary and exit
`-version`	Print version and exit
`-output <val>`	Name of the output file to create. The default output filename is ***ejoin-out.e***. If the specified file already exists, it will be overwritten.
`-omit_blocks <val>`	Omit the specified part/block pairs. The specification is `p#:block_id1:block_id2,p#:block_id1`. For example, to omit block ids 1,3,4 from part 1; block ids 2,3,4 from part 2; and block 8 from part5, specify `-omit_blocks p1:1:3:4,p2:2:3:4,p5:8`.
`-omit_nodesets [val]`	If no value, then don't transfer any nodesets to the output file. If just `p#,p#,...` specified, then omit nodesets on the specified parts. If `p#:id1:id2,p#:id2,id4...` then omit the nodesets with the specified id in the specified parts.
`-omit_sidesets [val]`	If no value, then don't transfer any sidesets to the output file. If just `p#,p#,...` specified, then omit sidesets on specified parts If `p#:id1:id2,p#:id2,id4...` then omit the sidesets with the specified id in the specified parts.
`-convert_nodal_to_nodesets <val>`	For each part listed (or `ALL`), create a nodeset containing the nodes of that part and output the nodal fields from that part as nodeset fields instead of nodal fields. Format is comma-separated list of parts (1-based), or `ALL`
`-match_node_ids`	Combine nodes if their global ids match. This option can not be specified if `-match_node_coordinates` is specified.
`-match_node_coordinates`	Combine nodes if they are within tolerance distance of each other. The `-tolerance <val>` option is used to set the matching tolerance. This option can not be specified if `-match_node_ids` is specified.
`-tolerance <val>`	Maximum distance between two nodes to be considered colocated. The default tolerance is calculated as 0.001 times the sum of the X, Y, and Z extents of the bounding box containing the overlapping regions of the bounding boxes of the two meshes being compared. This option is only meaningful if `-match_node_coordinates` is specified.
`-offset <val>`	Comma-separated x,y,z offset for the nodal coordinates of all parts except for part 1[1].
`-steps <val>`	Specify subset of timesteps to transfer to output file. Format is `beg:end:step`. For example, the input `-steps 1:10:2` would result in steps 1, 3, 5, 7, and 9 being transferred to the output file. To only transfer last step, use `-steps LAST`
`-gvar <val>`	Comma-separated list of global variables to be joined or `ALL` or `NONE`. The default behavior is that all global variables will be written to the output file.

[1]This option is planned to be changed in the future to allow specifying a different offset for each part including part 1. It is unknown at this time when this will be implemented or what the syntax will be.

`-evar <val>`	Comma-separated list of element variables to be joined or `ALL` or `NONE`. Variables can be limited to certain blocks by appending a colon followed by the block id. For example, `-evar sigxx:10:20` would result in the variable `sigxx` would be written for element blocks with id 10 and 20. The default behavior is that all element variables will be written to the output file.
`-nvar <val>`	Comma-separated list of nodal variables to be joined or `ALL` or `NONE`. The default behavior is that all nodal variables will be written to the output file.
`-nsetvar <val>`	Comma-separated list of nodeset variables to be joined or `ALL` or `NONE`. Variables can be limited to certain sets by appending a colon followed by the nodeset id. E.g. -nsetvar temperature:10:20. The default behavior is that all nodeset variables will be written to the output file.
`-ssetvar <val>`	Comma-separated list of sideset variables to be joined or `ALL` or `NONE`. Variables can be limited to certain sets by appending a colon followed by the nodeset id. E.g. -ssetvar sigxx:10:20. The default behavior is that all sideset variables will be written to the output file.
`-disable_field_recognition`	Do not try to combine scalar fields into higher-order fields such as vectors or tensors based on the field suffix.
`-copyright`	Show copyright and license data.

When EJoin is invoked, it will parse the user-specified options from the command line and it will also see if the environment variable `EJOIN_OPTIONS` exists. If the variable exists, then it will be parsed first followed by the parsing of the options on the command line.

4.3 Examples

The output below shows the results of a sample run of ejoin. In this example, there are three springs (drive, gate, and shutter) which have been prestressed in three separate analyses. The results data for these analyses are in the ****.rs*** files. The analyst wants to join these prestress outputs back into the original mesh and then perform another analysis using the prestressed springs. The spring element blocks have ids 1, 2, and 3. Each of the individual spring prestress runs have 3 element blocks – a spring and two spring attachment points. To create the final model, the following steps need to be done:

- Remove the spring element blocks from the original mesh database (***assembly.g***)
- Remove the spring attachment points from each prestressed spring model.
  - Blocks 14 and 21 in the drive spring (part 2),
  - Blocks 17 and 31 in the gate spring (part 3), and
  - Blocks 72 and 61 in the shutter spring (part 4).
- Join the remaining blocks into a single model including the results data from only the last timestep of the spring models.

These steps are accomplished using the following syntax:

```
ejoin -omit_blocks p1:1:2:3,p2:14:21,p3:17:31,p4:72:61 -steps LAST
      -output assembly_final.g assembly.g drive.rs gate.rs shutter.rs
```

The output from executing this command is:

```
EJoin
        (A code for merging Exodus II databases; with or without results data.)
        (Version: 1.2.7) Modified: 2011/02/09
*** 50503 Nodes were merged/omitted.

Wrote step    1/1, time 3.6000e-02

 Database: assembly_final.g

 Number of coordinates per node       =        3
 Number of nodes                      =   274723
 Number of elements                   =   220160
 Number of element blocks             =       25
 Number of nodal point sets           =       27
 Number of element side sets          =        4

 Number of global variables           =      128
 Number of variables at each node     =       42
 Number of variables at each element  =       34
 Number of variables at each nodeset  =        0
 Number of variables at each sideset  =        0

 Number of time steps on the database =        1
******* END *******
```

4.4 Related Codes

EJoin provides some, but not all, of the functionality of the `gjoin` [2] code. The main differences are:

- `gjoin` does not handle transient results data on the input databases.
- `gjoin` can combine element blocks from multiple input databases into a single element block. Similarly for nodesets and sidesets.
- `gjoin` can independently rotate, scale, and offset each input database; EJoin can currently only offset the second and subsequent parts by the same value; however, it is planned to add additional part transformation (offset, scale, rotation) options to EJoin in the future.
- `gjoin` tends to use memory inefficiently which can limit the size of database that it can generate; EJoin should use memory more efficiently and allow the generation of much larger models.

Chapter 5

Conjoin

5.1 Introduction

A finite element analysis can often generate multiple results and/or restart databases due to model bulk data changes or restarting the analysis.

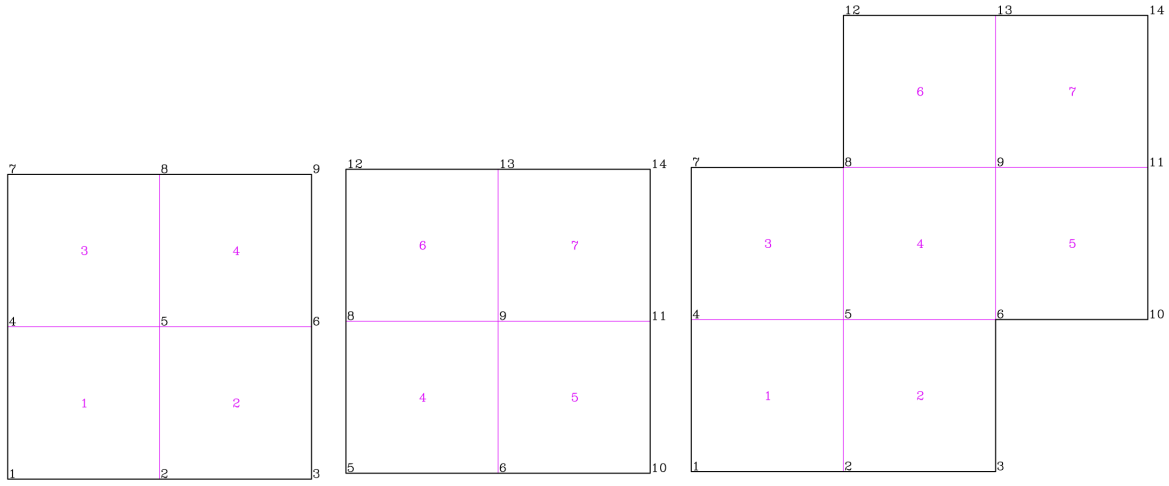
The bulk data changes are caused by element death, element creation, and surface evolution. Because the EXODUS database requires a constant model meta data and bulk data description, the analysis code must close the current results and restart databases and create new databases if the model meta data or bulk data change. At the end of the analysis, there will be multiple database files all describing a similar model with similar meta data, but with each database containing a different set of bulk data descriptions of that meta data.

Another cause of multiple EXODUS databases is when the analysis job is restarted one or more times. This is typically the case for long-running analyses whose total execution time exceeds the maximum allowed queue time on a compute cluster. A restart database is written at the end of each compute segment and then the job is resubmitted to the compute cluster and is restarted one or more times. At the end of the analysis, there will be at least one database for each compute segment and it is often desirable to combine these databases into a single database for visualization and post processing.

Conjoin joins two or more EXODUS databases into a single database. The input databases should represent the same model geometry with similar variables. The output database will contain the model geometry and all of the non-temporally-overlapping results data. If two databases have overlapping timestep ranges, the timesteps from the later database will be used. For example, if the first database contains time data from 0 to 5 seconds, and the second database contains time data from 4 to 10 seconds; the output database will contain time data from 0 to 4 seconds from the first database and time data from 4 to 10 seconds from the second database. If two nodes have the same global id and are also colocated, then they are combined to a single node in the output. Similarly, elements with the same global id and the same nodal connectivity are combined into a single element in the output file.

The output database will contain the union of the meta and bulk data entities (i.e., nodes, elements, element blocks, sidesets, and nodesets) from each input database. The existence of an entity at a particular timestep is indicated via a status variable. For example, if the first database is the mesh shown in Figure 5.1a at time 0.0 seconds, and the second database is the mesh shown in Figure 5.1b at time 1.0 seconds, then the combined mesh would be the union of the nodes and elements in the

two meshes as shown in Figure 5.1c.



(a) First Database, time 0 (b) Second Database, time 1 (c) Combined Database, time 0, 1

Figure 5.1: Example Input and Output Meshes for Conjoin

The values of the element status variable for each element at each timestep are shown below. A status value of 0 indicates that the element is active and a value of 1 indicates it is inactive. The nodes have a similar status variable.

element	1	2	3	4	5	6	7
status at time 0	0	0	0	0	1	1	1
status at time 1	1	1	1	0	0	0	0

The output database will contain the variables that are defined on the first mesh. If later databases do not have those variables, then the data will be zero-filled for those timesteps. If later databases have additional variables, they will be ignored.

5.2 Invoking Conjoin

The minimal command line for executing Conjoin is:

```
conjoin {list of files to join}
```

This would create an output mesh named ***conjoin-out.e*** containing the entities and data from the files listed on the command line. The files must be listed in order of increasing time step values. A nodal variable named `node_status` and an element variable named `elem_status` will be added to the variables already existing on the first file. The value of the status variables will be 0 to indicate an active node or element and 1 to indicate an inactive node or element.

Several options are available to modify the default behavior of Conjoin.

5.2.1 Options

`-help`	Print this summary and exit
`-output <val>`	Name of the output file to create. The default output filename is ***conjoin-out.e***. If the specified file already exists, it will be overwritten.
`-alive_value <val>`	Value (1 or 0) which will be used to indicate that an element is alive or active. The default behavior is that 0 indicates alive or active nodes and elements.
`-combine_status_variables <val>`	The conjoin elem_status variable will be combined with the specified status variable (val) existing on the mesh. Both variables must have the same value (1 or 0) to indicate that the element is alive or active. If 1 is alive, then the combined variable is the minimum of the two values. If 0 is alive, then the combined variable is the maximum of the two values. Use the `alive_value` option to set conjoin's alive value
`-element_status_variable <val>`	Element variable name to use as element existence status variable; it must not exist on input files. If specified as NONE, then it will not be written to the output file. Default = elem_status
`-nodal_status_variable <val>`	Nodal variable name to use as nodal status variable; it must not exist on input files. If specified as NONE, then it will not be written to the output file. Default = node_status
`-omit_nodesets`	Don't transfer nodesets to output file.
`-omit_sidesets`	Don't transfer sidesets to output file.
`-gvar <val>`	Comma-separated list of global variables to be joined or ALL or NONE. The default behavior is that all global variables will be written to the output file.
`-evar <val>`	Comma-separated list of element variables to be joined or ALL or NONE. Variables can be limited to certain blocks by appending a colon followed by the block id. For example, `-evar sigxx:10:20` would result in the variable `sigxx` would be written for element blocks with id 10 and 20. The default behavior is that all element variables will be written to the output file.
`-nvar <val>`	Comma-separated list of nodal variables to be joined or ALL or NONE. The default behavior is that all nodal variables will be written to the output file.
`-nsetvar <val>`	Comma-separated list of nodeset variables to be joined or ALL or NONE. The default behavior is that all nodeset variables will be written to the output file.
`-ssetvar <val>`	Comma-separated list of sideset variables to be joined or ALL or NONE. The default behavior is that all sideset variables will be written to the output file.

`-debug <val>`	Debug level (values are or'd) 1 = timing information. 4 = Verbose Element block information. 8 = Check consistent nodal coordinates between parts. 16 = Verbose Sideset information. 32 = Verbose Nodeset information. 64 = put exodus library into verbose mode. 128 = Check consistent global field values between parts.
`-width <val>`	Width of output screen, default = 80
`-version`	Print version and exit
`-copyright`	Show copyright and license data.

When Conjoin is invoked, it will parse the user-specified options from the command line and it will also see if the environment variable `CONJOIN_OPTIONS` exists. If the variable exists, then it will be parsed first followed by the parsing of the options on the command line.

5.3 Example

The output below shows the results of running the following command:

```
conjoin -output adapt.e adaptBox.e*
```

The first section of the output shows the code version followed by some information showing the map from part number to filename. In this case there are 6 parts which will be joined.

```
conjoin
        (A code for sequentially appending Exodus II databases. Supercedes conex and conex2.)
        (Version: 1.2.1) Modified: 2011/01/12
Part 1: 'adaptBox.e'
Part 2: 'adaptBox.e-s0002'
Part 3: 'adaptBox.e-s0003'
Part 4: 'adaptBox.e-s0004'
Part 5: 'adaptBox.e-s0005'
Part 6: 'adaptBox.e-s0006'
```

The next section of the output shows the output filename (***adapt.e*** in this example), and gives a summary of the output mesh showing the entity counts and the variable summary.

```
Output:   'adapt.e'
IO Word size is 8 bytes.
 Title: Default Sierra Title
```

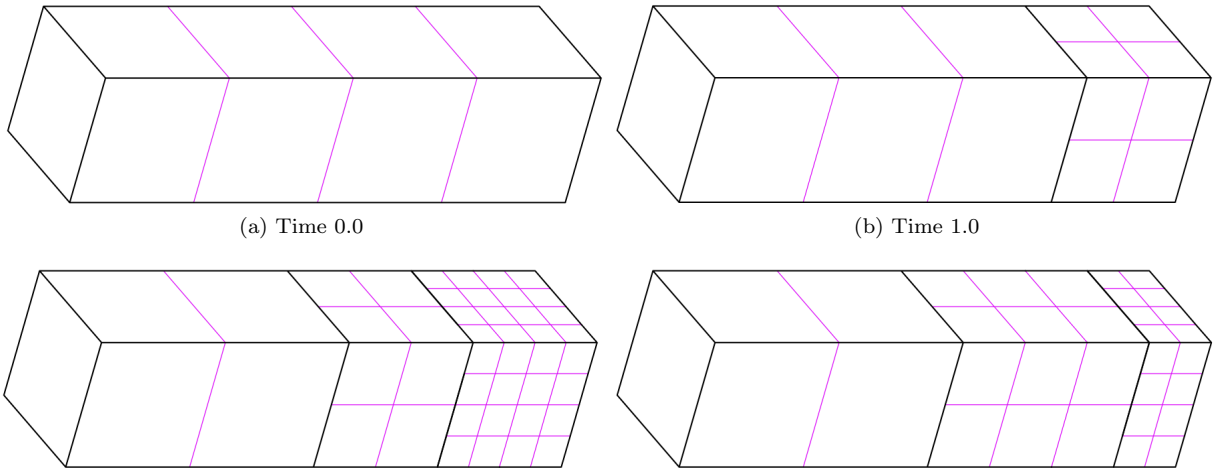
```
 Number of coordinates per node       =        3
 Number of nodes                      =      151
 Number of elements                   =       84
 Number of element blocks             =        1

 Number of nodal point sets           =        0
 Number of element side sets          =        2


Reading and Writing element connectivity & attributes

Wrote coordinate names...
Wrote coordinate information...
Found 7 global variables.
        ExternalEnergy  InternalEnergy  KineticEnergy   Momentum_x
        Momentum_y      Momentum_z      TIMESTEP

Found 7 nodal variables.
        displ_x      displ_y      displ_z      vel_x        vel_y
        vel_z        node_status

Found 7 element variables.
        stress_xx    stress_yy    stress_zz    stress_xy    stress_yz
        stress_zx    elem_status
```

The last section shows the progress of transferring the transient data to the output mesh. For each timestep, the output shows the part from which the data is being read and which timestep within that part is being used. It also show how many elements are active in that part and an estimate of how long it will take to finish transferring the data. Figures 5.2a to 5.2d show four views of the output mesh.

```
Step  1/11, time 0.0000e+00 (Part 1/6, step 1)  Active Elem:  4  [  9%, Elapsed=0.0s, ETA=0.0s]
Step  2/11, time 5.0000e-01 (Part 1/6, step 2)  Active Elem:  4  [ 18%, Elapsed=0.0s, ETA=0.0s]
Step  3/11, time 1.0000e+00 (Part 2/6, step 1)  Active Elem: 11  [ 27%, Elapsed=0.0s, ETA=0.0s]
Step  4/11, time 1.2500e+00 (Part 3/6, step 1)  Active Elem: 74  [ 36%, Elapsed=0.0s, ETA=0.0s]
Step  5/11, time 1.5000e+00 (Part 3/6, step 2)  Active Elem: 74  [ 45%, Elapsed= <1s, ETA= <1s]
Step  6/11, time 1.7500e+00 (Part 4/6, step 1)  Active Elem: 46  [ 55%, Elapsed= <1s, ETA= <1s]
Step  7/11, time 2.0000e+00 (Part 4/6, step 2)  Active Elem: 46  [ 64%, Elapsed= <1s, ETA= <1s]
Step  8/11, time 2.5000e+00 (Part 4/6, step 3)  Active Elem: 46  [ 73%, Elapsed= <1s, ETA= <1s]
Step  9/11, time 3.0000e+00 (Part 5/6, step 1)  Active Elem: 18  [ 82%, Elapsed= <1s, ETA= <1s]
Step 10/11, time 3.5000e+00 (Part 6/6, step 1)  Active Elem:  4  [ 91%, Elapsed= <1s, ETA= <1s]
Step 11/11, time 4.0000e+00 (Part 6/6, step 2)  Active Elem:  4  [100%, Elapsed= <1s, ETA=0.0s]
******* END *******
```

5.4 Related Codes

Conjoin replaces the functionality of the conex[3] code. Conjoin provides a superset of the conex functionality and is also much faster. Conjoin also supports nodeset and sideset variables and the



(a) Time 0.0

(b) Time 1.0

(c) Time 1.5

(d) Time 2.5

Figure 5.2: Example Output Mesh from Conjoin

naming of element blocks, nodesets, and sidesets which is not supported by `conex`. The input mesh databases to `Conex` are required to have the exact same bulk data which limits its use for adaptive analyses and other analyses which have varying numbers of active and inactive nodes and elements. `Conex` is no longer actively supported.

Bibliography

[1] Larry A. Schoof and Victor R. Yarberry, "EXODUSII: A Finite Element Data Model," SAND92-2137, Sandia National Laboratories, Albuquerque, NM, September, 1994.[1]

[2] Gregory D. Sjaardema, "GJOIN: A Program for Merging Two or More GENESIS Databases," SAND92-2290, Sandia National Laboratories, Albuquerque, NM, December 1992.

[3] Gregory D. Sjaardema, "CONEX: A code for sequentially appending ExodusII databases," unpublished internal communication, Sandia National Laboratories, Albuquerque, NM.

[4] Gary L. Hennigan, Matt M. St. John, Gregory D. Sjaardema, "NEM_JOIN: Join the results from a group of parallel ExodusII files," unpublished internal communication, Sandia National Laboratories, Albuquerque, NM.

[5] Gregory D. Sjaardema, "Overview of the Sandia National Laboratories Engineering Analysis Code Access System," SAND92-2292, Sandia National Laboratories, Albuquerque, NM, January 1993, Reprinted August 1994.

[1] This document is very out of date. A new document is being prepared and a draft of the current state is available at http://jal.sandia.gov/SEACAS/Documentation/exodusII-new.pdf.

Distribution

1 0899 Technical Library, 9536 (1 electronic)