Mapping EXODUS file data to the IOSS Model

Gregory D. Sjaardema

February 25, 2016

Contents

1 Introduction

The information contained in this document documents how the data in an EXODUS file are mapped into the standard IO subsystem (IOSS) model.

> **NOTE:** *This document assumes that the reader has familiarity with both the* EXODUS *API and the use of the IOSS; it is not a tutorial on either* EXODUS *or IOSS usage.*

An EXODUS file contains groupings of entity types such as nodes, edges, faces, and elements into either blocks or sets.

- A "block" grouping consists of homogenous entities of the same entity type and each entity is in one and only one block. The block has a method for identifying the topology of the contained entities.
- A "set" grouping contains entities of the same entity type (node, edge, face, or element); however, the topologies of the entities (quad vs. triangle face) do not need to be the same. An entity can be in multiple entity sets, but it is not required to be in any.

The finite element model data (sets, blocks, fields, etc.) described in the EXODUS file is stored in an Ioss::Region and its owned grouping entity classes. The table below shows the mapping between the EXODUS entity groups and the IOSS grouping entity classes. All of the IOSS classes are derived from the Ioss::GroupingEntity class:

	node	edge	face	element	"side"
set	Ioss::NodeSet	Ioss::EdgeSet	Ioss::FaceSet	Ioss::ElementSet	Ioss::SideSet
block	Ioss::NodeBlock	Ioss::EdgeBlock	Ioss::FaceBlock	Ioss::ElementBlock	

An EXODUS "side" is an entity type defined by a pair consisting of an element and a local side of that element. The EXODUS data model does not have a concept of a homogenous grouping (block) of "side" entity types; however,the IOSS will group the "sides" in an EXODUS side set into homogenous subsets with a class of Ioss::SideBlock. This is described in more detail in Section **??**

Each entity group in the EXODUS file will result in the creation of the corresponding Ioss::GroupingEntity class. All of these classes will be managed by a Ioss::Region which corresponds to the mesh model described by the EXODUS file.

In addition to the above classes, the optional communication data that might exist in the EXODUS file is mapped to node and elementIoss::CommSet object. If the EXODUS file contains one or more coordinate frames, then there will be a Ioss::CoordinateFrame object for each coordinate frame.

Only node blocks and sets, element blocks, side sets, communication sets, and coordinate frames are discussed below. The remaining types are supported by IOSS, but are not used as often. Their functionality and capabilities are very similar to the types discussed below.

2 Nodes

Node Grouping

A single nodeblock named "nodeblock_1" will be created for the mesh. It contains information for every node that exists in the model. An additional grouping of nodes is possible via EXODUS node sets which map to Ioss::NodeSets which are described in Section 4.

Node IDS

- The global ids of the nodes are stored in the field *"ids"* which is of type scalar. The data will be either 32-bit or 64-bit integers depending on the type stored in the EXODUS file.
- The global ids are read from the database as follows:
  - If there exists a node map with the name "original_global_id_map" on the EXODUS file, then it is used to define the global ids.
  - Otherwise, the global ids are obtained from the values returned from the ex_get_id_map EXODUS API function.
- Relevant EXODUS API function(s): ex_get_num_map, ex_get_id_map.

Node Coordinates

- The nodal coordinates are accessed via the field *"mesh_model_coordinates"* which is a double field with "number of spatial dimensions" components. For a 3D model, the x,y,z coordinates of the first node are stored followed by the x,y,z coordinates of the next node. For a 2D model, only the x and y coordinates are stored; a 1D model has only the x coordinates. The order of the nodes matches the order specified in the *"ids"* field.

- An individual component of the nodal coordinates can be accessed via the fields:
  - *"mesh_model_coordinates_x"*
  - *"mesh_model_coordinates_y"* (only exists for 2D or 3D models)
  - *"mesh_model_coordinates_z"* (only exists for 3D models)

- Relevant EXODUS API function(s): ex_get_coord.

Node Attributes

- If the nodes have attributes, they are added as Ioss::Fields with role Ioss::Field::ATTRIBUTE[1].

- There will always be a field named *"attribute"*. The number of components in the field will be equal to the number of nodal attributes. This field can be used to access the data for all attributes.

- In addition, if the EXODUS file contains attributes that are named, then there will be a field for each attribute or "combined attribute" (See Section 9). If the attribute names indicate that they are the components of a "higher-order" type then the field will be of that type.

- Ioss::Field has a member function get_index() which returns the 1-based index of this attribute field in the original EXODUS ordering of the attributes. This should not be used unless really needed, but is provided since some EXODUS-based applications do not support named attributes and instead rely on the implicit ordering of the attributes in the EXODUS file. Note that both the *"attribute"* field containing all attributes and the first named attribute will have an index value of 1. Another method for accessing the attributes in "file order" is shown below.

- The names of all fields with role "attribute" defined on the Ioss::NodeBlock can be retrieved with the call field_describe(Ioss::Field::ATTRIBUTE, &results_fields) where `results_fields` is a container of type Ioss::NameList which is `std::vector<std::string>`

- If the EXODUS file contains named attributes, but the analysis code does not want to use those names, the database property *"IGNORE_ATTRIBUTE_NAMES"* can be set prior to reading the EXODUS file. In that case, the attributes will generate #attribute scalar fields named: `attribute_1`, `attribute_2`, ..., `attribute_#`

- Relevant EXODUS API function(s): ex_get_attr, ex_get_attr_param, ex_get_attr_names.

[1]The "role" of an Ioss::Field indicates how the field is used. Valid values for role are INTERNAL, MESH, ATTRIBUTE, COMMUNICATION, INFORMATION, REDUCTION, and TRANSIENT

Node Fields

If there are any nodal transient fields on the mesh database, then there will be an Ioss::Field with role Ioss::Field::TRANSIENT defined for "each" field. If the naming of the field indicates that the fields are components of a higher-order field (See Section 9) then the Ioss::Field will be of that type.

- Relevant EXODUS API function(s): ex_get_variable_name, ex_get_variable_names, ex_get_var, ex_get_variable_param, ex_get_truth_table.

3 Elements

Element Grouping (Element Blocks)

Each element block on the EXODUS file will be mapped into an Ioss::ElementBlock. In addition to the normal Ioss::ElementBlock properties, the following EXODUS-specific properties will also be defined. Note that although these are defined, relying on their values in an application code precludes the use of a non-EXODUS database type which may not provide a similar property.

Property	Description
id	The integer element block id
original_block_order	The 0-based order of the element block description in the EXODUS file.
original_topology_type	The element type string specified for the element block.

Element Block Names

An element block can have multiple names. There will be one canonical name that is returned by the Ioss::ElementBlock::name() function call; additional names are referred to as aliases.

- If the EXODUS file has a name for the element block, then that name will be the canonical name of the Ioss::ElementBlock.
- If this name is not all lowercase, then an alias of the lowercased name will be created.
- An additional name is constructed by concatenating the string "block_" with the numerical block id. This will be the canonical name if the file does not have explicit names for the blocks.

For example, if the element block with id '10' on the EXODUS file is named 'FireSet', The corresponding Ioss::ElementBlock will have the canonical name 'FireSet.' That element block will also have the aliases 'fireset', and 'block_10' defined. If the element block with id '123' has no name, then it will have the canonical name 'block_123' assigned to it.

- Relevant EXODUS API function(s): ex_get_name, ex_get_names.

Element IDS

The global ids of all elements are stored in the field *"ids"* which is of type scalar. The data will be either 32-bit or 64-bit integers depending on the type stored in the EXODUS file.

The global ids are read from the database as follows:

- If there exists a element map with the name "original_global_id_map" on the EXODUS file, then it is used to define the global ids
- Otherwise, the global ids are obtained from the values returned from the ex_get_id_map EXODUS API function.
- Relevant EXODUS API function(s): ex_get_num_map, ex_get_id_map.

Element Connectivity

- The connectivity of the elements in a block is returned via the field *"connectivity"*. It will return a field with #nodes_per_element components. The nodal ids in the connectivity will be global ids.
- The "raw" connectivity of the elements in a block is returned via the field *"connectivity_raw"*. Here "raw" is defined such that a raw id N refers to the Nth node (1-based) in the list of ids returned by the nodeblock *"ids"* field. For example, if the nodeblock ids field contains "10,20,30,40", then a raw id of '1' refers to the same node as the node with global id "10". The range of the "raw" ids is 1..#entity_on_processor
- Relevant EXODUS API functions: ex_get_conn

Element Topology

The EXODUS "element type" is mapped to an Ioss::ElementTopology class. The type is converted to lower case, multiple embedded spaces are collapsed to a single space, and then all spaces are replaced with an underscore. If the type name does not end with a number, then the "number of nodes per element" is appended to the end of the name. For example, a 20-node hex element block with the type name "hex" would be converted to the name "hex20" before being mapped to an Ioss::ElementTopology. The mapping is as follows:

IOSS	EXODUS element type string.
bar2	rod_2_2d, rod_2_3d, bar, beam, beam-r, beam-r2, beam2, beam_2, line, line2, rod, rod2, rod2d2, rod3d2, rod_2_2d, rod_2_3d, truss, truss2, shell2(2D)
bar3	beam_3, rod_3_2d, rod_3_3d, beam3, rod2d3, rod3, rod3d3, rod_3_2d, rod_3_3d, truss3, shell3(2D)
hex20	hexahedron_20, solid_hex_20_3d, solid_hex_20_3d
hex27	hexahedron_27, solid_hex_27_3d, solid_hex_27_3d
hex8	hexahedron_8, solid_hex_8_3d, hex, solid_hex_8_3d

pyramid13	pyramid_13, solid_pyramid_13_3d, pyra13, solid_pyramid_13_3d
pyramid14	pyramid_14, solid_pyramid_14_3d, pyra14, solid_pyramid_14_3d
pyramid5	pyramid_5, solid_pyramid_5_3d, pyra5, pyramid, solid_pyramid_5_3d
quad4	face_quad_4_3d, quadrilateral_4_2d, quadrilateral_4, solid_quad_4_2d, quad, quadface4, quadrilateral_4, quadrilateral_4_2d, solid_quad_4_2d
quad8	face_quad_8_3d, quadrilateral_8_2d, quadrilateral_8, solid_quad_8_2d, quadface8, quadrilateral_8, quadrilateral_8_2d, solid_quad_8_2d
quad9	face_quad_9_3d, quadrilateral_9_2d, quadrilateral_9, solid_quad_9_2d, quadface9, quadrilateral_9, quadrilateral_9_2d, solid_quad_9_2d
shell4	shell_quadrilateral_4, shellquadrilateral_4, shell_quad_4_3d, shell, shell_quad_4_3d, shellquadrilateral_4
shell8	shell_quadrilateral_8, shellquadrilateral_8, shell_quad_8_3d, shell_quad_8_3d, shellquadrilateral_8
shell9	shell_quadrilateral_9, shellquadrilateral_9, shell_quad_9_3d, shell_quad_9_3d, shellquadrilateral_9
shellline2d2	shell_line_2, shellline_2, shell_line_2_2d, shell_line_2_2d, shellline_2, shell2
shellline2d3	shell_line_3, shellline_3, shell_line_3_2d, shell_line_3_2d, shellline_3, shell3
sphere	particle, particle_1_2d, particle_1_3d, circle(2D), circle1(2D), particle_1_2d, particle_1_3d, particles, point, point1, sphere-mass, sphere, sphere1
tetra10	solid_tet_10_3d, tetrahedron_10, tet10, tetrahedron_10
tetra11	solid_tet_11_3d, tetrahedron_11, tet11, tetrahedron_11
tetra4	solid_tet_4_3d, tetrahedron_4, tet4, tetra, tetrahedron_4
tetra8	solid_tet_8_3d, tetrahedron_8, tet8, tetrahedron_8
tri3	face_tri_3_3d, solid_tri_3_2d, triangle_3_2d, triangle_3, solid_tri_3_2d, tri, triangle(2D), triangle3(2D), triangle_3, triangle_3_2d, triface3
tri4	face_tri_4_3d, solid_tri_4_2d, triangle_4_2d, triangle_4, solid_tri_4_2d, triangle4(2D), triangle_4, triangle_4_2d, triface4
tri6	face_tri_6_3d, solid_tri_6_2d, triangle_6_2d, triangle_6, solid_tri_6_2d, triangle6(2D), triangle_6, triangle_6_2d, triface6
trishell3	shell_triangle_3, shelltriangle_3, shell_tri_3_3d, shell3(3D), shell_tri_3_3d, shelltriangle_3, trishell, triangle(3D), triangle3(3D)

trishell4	shell_triangle_4, shelltriangle_4, shell_tri_4_3d, shell_tri_4_3d, shelltriangle_4, triangle4(3D)
trishell6	shell6, shell_triangle_6, shelltriangle_6, shell_tri_6_3d, shell_tri_6_3d, shell_triangle_6, shelltriangle_6, triangle6(3D)
unknown	invalid_topology
wedge15	solid_wedge_15_3d, wedge_15, wedge_15
wedge18	solid_wedge_18_3d, wedge_18, wedge_18
wedge6	solid_wedge_6_3d, wedge_6, wedge_6, wedge, wedge_6
super#	super, superelement

A superelement type is basically a collection of 0 or more nodes that are treated in a special way by certain applications. In IOSS, a superelement type is mapped to a Ioss::ElementTopology named "super#" where "#" is replaced by the number of nodes in the superelement's connectivity. For example, a superelement with 42 nodes would be mapped to the Ioss::ElementTopology "super42"

- Relevant EXODUS API function(s): ex_get_block, ex_get_block_param, ex_get_elem_type

Element Attributes

If the elements in an element block have attributes, the attributes are added as Ioss::Fields with role Ioss::Field::ATTRIBUTE.

There will always be a field named *"attribute"*. The number of components in the field will be equal to the number of attributes on the element block. This field can be used to access the data for all attributes. In addition:

- If the attributes are named, then there will be a field for each attribute or "combined attribute". If the attribute names indicate that they are the components of a "higher-order" type (e.g. vector or tensor) then the field will be of that type. See Section 9 for details on how the scalar EXODUS fields are combined into higher-order storage types.

- If the attributes are not named, then the names will be inferred via EXODUS and application conventions. The conventions used by IOSS are:

Element Type	Index	Inferred Attribute Name	
circle or sphere	1	*"radius"*	scalar
	2	*"volume"*	scalar
sphere-mass	1	*"mass"*	scalar
	2-7	*"inertia"*	symmetric tensor
	8,9,10	*"offset"*	3D vector
truss, bar, beam, rod	1	*"area"*	scalar
(2D)	2,3	*"i", "j"*	scalar
(3D)	2,3,4	*"i1", "i2", "j"*	scalar
	5,6,7	*"reference_axis"*	3D vector
	8,9,10	*"offset"*	3D vector
shell,trishell	1	*"thickness"*	scalar
shell,trishell	1..#node	*"nodal_thickness"*	REAL[#node]

- For the shell and trishell element types, if there is only a single attribute, then it is interpreted as the *"thickness"*; if the attribute count matches the number of nodes in the element connectivity, then the attribute is interpreted as the *"nodal_thickness"*.
- If the element block's attribute count is smaller than what is listed above, only the attributes provided will be assigned to a field. For example, if a "circle" element block has only 1 attributed, only the *"radius"* field will be defined.
- Additional unrecognized attributes would be accessible via a field named *"extra_attribute_#"* where the "#" is replaced by the number of unrecognized attributes. This will be a single field with "#" components and a storage type of "Real[#]".

An Ioss::Field has a member function get_index() which will return the 1-based index of this attribute field in the original EXODUS ordering of the attributes. This should not be used unless really needed. Note that both the *"attribute"* field containing all attributes and the first named attribute will have an index value of 1.

The names of all fields with role "attribute" defined on an element block can be retrieved with the call field_describe(Ioss::Field::ATTRIBUTE, &results_fields) where `results_fields` is a container of typeIoss::NameList which is `std::vector<std::string>`

If the EXODUS file contains named attributes, but the analysis code does not want to use those names, the database property *"IGNORE_ATTRIBUTE_NAMES"* can be set prior to reading the EXODUS file. In that case, the attributes will generate #attribute scalar fields named: "attribute_1", "attribute_2", ..., "attribute_#"

- Relevant EXODUS API function(s): ex_get_attr, ex_get_attr_param, ex_get_attr_names.

Element Fields

If there are any element transient fields on the mesh database, then there will be an Ioss::Field with role Ioss::Field::TRANSIENT defined for "each" field. If the naming of the field indicates that the fields are components of a higher-order field, then the Ioss::Field will be of that type. See Section 9 for additional details.

- Relevant EXODUS API function(s): ex_get_variable_name, ex_get_variable_names, ex_get_var, ex_get_variable_param, ex_get_truth_table.

4 Node Sets

Each node set on the EXODUS file will be mapped into an Ioss::NodeSet. In addition to the normal Ioss::NodeSet properties, the following EXODUS-specific properties will also be defined. Note that although these are defined, relying on their values in an application code precludes the use of a non-EXODUS database type which may not provide a similar property.

Property	Description
id	The integer node set id

Node Set Names

A node set can have multiple names. There will be one canonical name that is returned by the name() function call; additional names are referred to as aliases.

- If the EXODUS file has a name for a node set, then that name will be the canonical name of the Ioss::NodeSet
- If this name is not all lowercase, then an alias of the lowercased name will be created.
- An additional name is constructed by concatenating the string "nodelist_" with the numerical node set id. This will be the canonical name if the file does not have explicit names for the node sets.
- An additional name is constructed by concatenating the string "nodeset_" with the numerical sideset id.
- Relevant EXODUS API functions: ex_get_name, ex_get_names.

Distribution Factors

A *"distribution_factors"* field will be defined for all Ioss::NodeSets whether or not they are defined on the EXODUS file. If they do not exist on the file, the values returned by the field will be equal to 1.0; otherwise, they will be the values stored on the file.

Node Set Attributes

- If the nodes in the node set have attributes, the attributes are added as Ioss::Fields with role Ioss::Field::ATTRIBUTE.
- There will always be a field named *"attribute"*. The number of components in the field will be equal to the number of node set attributes. This field can be used to access the data for all attributes.

- In addition, if the EXODUS file contains attributes that are named, then there will be a field for each attribute or "combined attribute" (See Section 9). If the attribute names indicate that they are the components of a "higher-order" type then the field will be of that type.

- Ioss::Field has a member function get_index() which returns the 1-based index of this attribute field in the original EXODUS ordering of the attributes. This should not be used unless really needed, but is provided since some EXODUS-based applications do not support named attributes and instead rely on the implicit ordering of the attributes in the EXODUS file. Note that both the *"attribute"* field containing all attributes and the first named attribute will have an index value of 1. Another method for accessing the attributes in "file order" is shown below.

- The names of all fields with role "attribute" defined on the Ioss::NodeSet can be retrieved with the call field_describe(Ioss::Field::ATTRIBUTE, &results_fields) where `results_fields` is a container of type Ioss::NameList which is `std::vector<std::string>`

- If the EXODUS file contains named attributes, but the analysis code does not want to use those names, the database property *"IGNORE_ATTRIBUTE_NAMES"* can be set prior to reading the EXODUS file. In that case, the attributes will generate #attribute scalar fields named: `attribute_1`, `attribute_2`, ..., `attribute_#`

- Relevant EXODUS API function(s): ex_get_attr, ex_get_attr_param, ex_get_attr_names.

Node Set Fields

If there are any node set transient fields on the mesh database, then there will be an Ioss::Field with role Ioss::Field::TRANSIENT defined for "each" field. If the naming of the field indicates that the fields are components of a higher-order field, then the Ioss::Field will be of that type. See Section 9 for additional details.

- Relevant EXODUS API function(s): ex_get_variable_name, ex_get_variable_names, ex_get_var, ex_get_variable_param, ex_get_truth_table.

5 Sides

Side Grouping (Side Sets)

Each sideset on the EXODUS file will be mapped into an Ioss::SideSet. In addition to the normal Ioss::SideSet properties, the following EXODUS-specific properties will also be defined. Note that although these are defined, relying on their values in an application code precludes the use of a non-EXODUS database type which may not provide a similar property.

Property	Description
id	The integer side set id

Side Set Names

A side set can have multiple names. There will be one canonical name that is returned by the name() function call; additional names are referred to as aliases.

- If the EXODUS file has a name for the sideset, then that name will be the canonical name of the Ioss::SideSet
- If this name is not all lowercase, then an alias of the lowercased name will be created.
- An additional name is constructed by concatenating the string "surface_" with the numerical sideset id. This will be the canonical name if the file does not have explicit names for the sidesets.
- An additional name is constructed by concatenating the string "sideset_" with the numerical sideset id.
- Relevant EXODUS API functions: ex_get_name, ex_get_names.

SideBlocks

The IOSS examines each of the element/local side pairs in the EXODUS side set side list and combines then into homogenous groups which are called Ioss::SideBlocks. A Ioss::SideSet consists of 1 or more Ioss::SideBlock. There are a few options on how the element/local_side pairs in an EXODUS sideset are categorized into homogenous Ioss::SideBlock's. The categorization is controlled by calling the set_surface_split_type() function on the Ioss::DatabaseIO pointer. The argument to this function is of type Ioss::SurfaceSplitType. The valid values are their function are:

SPLIT_BY_DONT_SPLIT create only a single sideblock containing all pairs.

SPLIT_BY_TOPOLOGIES all local element sides with the same topology and underlying element topology are put in the same side block. For example 'quad4' sides on 'hex8' elements.

SPLIT_BY_ELEMENT_BLOCK all local element sides with the same topology and the elements are in the same element block are put in the same side block. For example 'quad4' sides on 'hex8' elements. For example all 'tri3' faces on elements in 'my_tet_block'.

The default setting is SPLIT_BY_TOPOLOGIES.

The names of the Ioss::SideBlocks in a Ioss::SideSet will be "surface_" + the base element topology or the base element block name + "_" + side topology name + "_" + sideset id. For example, a Ioss::SideBlock containing quad4 faces on hex8 elements in a Ioss::SideSet with id 10 would be named "surface_hex8_quad4_10". Note that the topology information in the name can be obtained from the Ioss::SideBlock object itself instead of decoding the name, so don't try to decode the name to get the information.

Distribution Factors

A *"distribution_factors"* field will be defined for all Ioss::SideBlocks whether or not they are defined on the EXODUS file. If they do not exist on the file, the values returned by the field will be equal to 1.0; otherwise, they will be the values stored on the file. Note that since a Ioss::SideBlock contains element sides of a homogenous topology, the number of distribution factors is easy to determine from the topology of the Ioss::SideBlock.

Side Set Attributes

- If the sides in the side set have attributes, the attributes are added as Ioss::Fields with role Ioss::Field::ATTRIBUTE.
- There will always be a field named *"attribute"*. The number of components in the field will be equal to the number of node set attributes. This field can be used to access the data for all attributes.
- In addition, if the EXODUS file contains attributes that are named, then there will be a field for each attribute or "combined attribute" (See Section 9). If the attribute names indicate that they are the components of a "higher-order" type then the field will be of that type.
- Ioss::Field has a member function get_index() which returns the 1-based index of this attribute field in the original EXODUS ordering of the attributes. Note that both the *"attribute"* field containing all attributes and the first named attribute will have an index value of 1. This should not be used unless really needed, but is provided since some EXODUS-based applications do not support named attributes and instead rely on the implicit ordering of the attributes in the EXODUS file. Another method for accessing the attributes in "file order" is shown below.
- The names of all fields with role "attribute" defined on the Ioss::SideBlock can be retrieved with the call field_describe(Ioss::Field::ATTRIBUTE, &results_fields) where `results_fields` is a container of type Ioss::NameList which is `std::vector<std::string>`
- If the EXODUS file contains named attributes, but the analysis code does not want to use those names, the database property *"IGNORE_ATTRIBUTE_NAMES"* can be set prior to reading the EXODUS file. In that case, the attributes will generate #attribute scalar fields named: `attribute_1`, `attribute_2`, ..., `attribute_#`
- Relevant EXODUS API function(s): ex_get_attr, ex_get_attr_param, ex_get_attr_names.

Side Set Fields

If there are any side set transient fields on the mesh database, then there will be an Ioss::Field with role Ioss::Field::TRANSIENT defined for "each" field. If the naming of the field indicates that the fields are components of a higher-order field, then the Ioss::Field will be of that type. For example, EXODUS fields named d_x, d_y, d_z would define an Ioss::Field of type VECTOR_3D named "d". See Section 9 for additional details. The fields will be defined on the Ioss::SideBlock and not on the Ioss::SideSet.

- Relevant EXODUS API function(s): ex_get_variable_name, ex_get_variable_names, ex_get_var, ex_get_variable_param, ex_get_truth_table.

6 Parallel Communication Data

The EXODUS communication map information is mapped into Ioss::CommSet objects. In a parallel execution, there will be two Ioss::CommSet objects; one for nodal communication data named "commset_node" and one for element communication data named "commset_side". The groups managed by the Ioss::CommSet are the list of nodes or element sides that are on the current processor and are also on another processor; these are often referred to as "shared node" or "shared sides". The defined fields are *"entity_processor"* and *"entity_processor_raw"*. The field data consists of pairs of entity ids and the processor that this entity is shared with; for a "commset_side" group, the "entity id" is an "element/1-based local face index" pair. For the *"entity_processor"*, the entity ids are global ids; for the *"entity_processor_raw"*, the entity ids are "raw" ids. Here "raw" is defined such that a raw id N refers to the Nth node (1-based) in the list of ids returned by the nodeblock *"ids"* field. For example, if the nodeblock ids field contains "10,20,30,40", then a raw id of '1' refers to the same node as the node with global id "10". The range of the "raw" ids is 1..#entity_on_processor.

7 Coordinate Frames

- Each coordinate frame on the EXODUS file will create a Ioss::CoordinateFrame which is identified by its id.
- A coordinate frame can be retrieved from the Ioss::Region via the get_coordinate_frame(id) function.
- All coordinate frames can be retrieved from the Ioss::Region via the get_coordinate_frames() which returns a reference to a Ioss::CoordinateFrameContainer.
- A coordinate frame has the functions:
  - id() - returns the id;
  - tag() - returns the character tag. The tag will be 'r' for rectangular, 'c' for cylindrical, or 's' for spherical. However, the tag is not restricted to these values and whatever value was in the EXODUS file will be returned.
  - coordinates() – returns all 9 values stored for a coordinate frame.
  - origin() – returns the 3 points defining the origin of the coordinate frame.
  - axis_3_point() – returns the 3 values defining a point on axis 3.
  - plane_1_3_point() – returns the 3 values defining a point on the 1-3 coordinate plane

The coordinate frame definition is not checked for validity to ensure that it defines a valid three-dimension coordinate system.

- Relevant EXODUS API functions: ex_get_coordinate_frames

8 Quality Assurance and Information Records

The Quality Assurance (QA) records in an EXODUS file can be accessed as a C++ reference to a `std::vector<std::string>` via the get_qa_records() member function on the Ioss::Region. The strings are interpreted in groups of four corresponding to the similar grouping in the EXODUS api. The records are interleaved in the order:

1. Code Name
2. Code Version
3. Execution Date
4. Execution Time

For each record. If the application does not need to access the existing QA records, but just wants to add its own information to the QA records, there is a member function add_qa_record. This function takes four `std::string` arguments in the same order as shown above. The date and time arguments are optional and if omitted, the current date and time will be used. The strings will be truncated to a maximum of 32 characters each to meet the EXODUS requirement.

In a similar matter, the EXODUS "Informational Records" can be accessed as a C++ reference to a `std::vector<std::string>` via the get_information_records member function on the Ioss::Region. If the application only needs to add additional information records to those that already exist on the input mesh (for output to a results file, for example), it can use the add_information_record to add a single information line, or the add_information_records to add multiple information lines. Both functions are defined on the Ioss::Region class; the first takes a `std::string` argument and the second takes a `std::vector<std::string>` argument. The string(s) will be truncated to a maximum of 80 characters to meet the EXODUS requirement.

- Relevant EXODUS API function(s): ex_get_qa, ex_get_info, ex_put_qa, ex_put_info

9 Attribute and Variable Mapping to Ioss::Field

Each transient variable on the EXODUS file will be mapped to a field on the IOSS object corresponding to the EXODUS entity with that variable. The names of the EXODUS variables are examined and if the IOSS can combine the names into a "higher-order" type such as a vector, tensor, symmetric tensor, or other type, then it will create a field corresponding to that higher-order type. The mapping is done by splitting the variable name into a "base name" and a "suffix" separated by the "field suffix separator" character. The default field suffix separator is the underscore "_", but it can be changed via the Ioss::DatabaseIO::set_field_separator(new_separator) function. Once the names are separated into base name and suffix, then all names with the same base name are examined to see if their corresponding suffices indicate components of a higher-order type.

For example, if the variable names are "d_x", "d_y", "d_z", and "v_x", then the suffices of the three variables with the base name "d" would be examined. These are "x", "y", and "z" which match the suffices in the Ioss::VariableType class corresponding to a 3D Vector type. In this instance, the IOSS would add a field "d" to the entity and the field would be of type Ioss::Vector_3D.

The ***Ioss_ ConcreteVariableType.C*** file lists the suffices that are examined and the IOSS variable type classes.

An additional grouping of variables is done if the suffices form a sequence of integers (1,2,...). In this case, a Ioss::VariableType named "Real[#]" is created with "#" equal to the maximum integer in the sequence. If the count exceeds 10, then the suffices must all be of the same width, so for example a 12 component field would have suffices "01", "02", ..., "12".

The mapping of scalar EXODUS fields into a higher-order type can be disabled by setting the field suffix separator to a space (" "). If the separator is set to the value 0 (not the character '0'), then the IOSS code finds a group of names that share the maximum length of base name and then treats the remainder of the name as the suffix and then does the same mapping as above. For example, the fields "velocityx", "velocityy", "velocityz" would isolate the base name "velocity" with suffices "x", "y", "z" and create a field named "velocity" of type Ioss::Vector_3D.

This mapping is done for both EXODUS transient variables and for EXODUS attributes.