SANDIA REPORT

SAND2016-9964
Unlimited Release
Printed September 20, 2016

Sierra Toolkit Manual Version 4.42

Sierra Toolkit Development Team

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

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. 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

SAND2016-9964
Unlimited Release
Printed September 20, 2016

Sierra Toolkit Manual Version 4.42

Sierra Toolkit Development Team

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185

Abstract

This report provides documentation for the SIERRA Toolkit (STK) modules. STK modules are intended to provide infrastructure that assists the development of computational engineering software such as finite-element analysis applications. STK includes modules for unstructured-mesh data structures, reading/writing mesh files, geometric proximity search, and various utilities. This document contains a chapter for each module, and each chapter contains overview descriptions and usage examples. Usage examples are primarily code listings which are generated from working test programs that are included in the STK code-base. A goal of this approach is to ensure that the usage examples will not fall out of date.

This page intentionally left blank.

Contents

1 STK Mesh **17**

1.1 STK Mesh Terms 17

1.1.1 Entity 18

1.1.2 Connectivity 18

1.1.3 Topology 18

1.1.4 Part 19

1.1.5 Field 19

1.1.6 Selector 19

1.1.7 Bucket 20

1.1.8 Ghosting 20

1.1.9 MetaData and BulkData 20

1.1.10 Creating a STK Mesh from an Exodus file 22

1.2 Parallel 22

1.2.1 Shared 23

1.2.2 Ghosted 23

1.2.3 Aura 23

1.2.3.1 How to use automatically generated aura 23

1.3 STK Parallel Mesh Consistency Rules 24

1.3.1 How to enable mesh diagnostics to enforce parallel mesh rules 25

1.3.2 How to enforce Parallel Mesh Rule 1 26

1.3.3 Parallel API 26

1.4 STK Mesh Selector 28
1.4.1 How to use selectors 29
1.5 STK Mesh Parts 30
1.5.1 Part Identifiers and Attributes 31
1.5.2 Induced Part Membership 32
1.5.3 How to use ghost parts 33
1.6 Mesh Modification 35
1.6.1 Overview 35
1.6.2 Public Modification Capability 36
1.6.2.1 Add/Delete Entities 36
1.6.2.2 Getting Unused Globally Unique Identifiers 37
1.6.2.3 Creating Nodes that are Shared by Multiple Processors 38
1.6.2.4 Change Entity Part Membership 41
1.6.2.5 Change Connectivity 41
1.6.2.6 Change Entity Ownership 42
1.6.2.7 Change Ghosting 42
1.6.3 Mesh Modification Examples 42
1.6.3.1 Resolving Sharing Of Exodus Sidesets - Special Case 44
1.6.4 Unsafe operations 46
1.6.5 Automatic modification operations in modification_end() 47
1.6.6 How to use generate_new_entities() 47
1.6.7 How to create faces 49
1.6.8 How to create both edges and faces 49
1.6.9 How to create faces on only selected elements 50
1.6.10 Creating faces with layered shells 51
1.6.11 Creating faces between hexes, on shells, and on shells between hexes 52

1.6.12 How to skin a mesh 54
1.6.13 How to create internal block boundaries of a mesh 55
1.6.14 How to destroy elements in list 56
1.7 STK Mesh usage examples 56
1.7.1 How to iterate over nodes 56
1.7.2 How to traverse connectivity 58
1.7.3 How to check side equivalency 60
1.7.4 Understanding node ordering of edges and faces 60
1.7.5 How to sort entities into an arbitrary order 61

2 STK Topology **63**

2.1 STK Topology API 63
2.1.1 How to set and get topology 64
2.1.2 STK topology ranks 64
2.1.3 Compile-time STK topology information 66
2.1.4 STK topology for the Particle 66
2.1.5 STK topology for the high order Beam 67
2.1.6 STK topology for the high order triangular Shell 68
2.1.7 STK topology for the linear Hexahedral 69
2.1.8 STK topology **`equivalent`** method 71
2.1.9 STK topology's **`is_positive_polarity`** method 72
2.1.10 STK topology's **`lexicographical_smallest_permutation`** method 72
2.1.11 STK topology's **`lexicographical smallest permutation preserve polarity`** method 73
2.1.12 STK Topology's **`sub_topology`** methods 74
2.1.13 STK Topology's **`sides`** methods 75

2.1.14 STK topology for a SuperElement 75
2.2 Mapping of Sierra topologies 76

3 STK Fields **81**
3.1 Example STK fields usage 81

4 STK IO **87**
4.1 STK IO: usage examples 87
4.1.1 Reading mesh data to create a STK Mesh 87
4.1.1.1 Face creation for input sidesets 88
4.1.2 Reading mesh data to create a STK Mesh allowing StkMeshIoBroker to go out of scope 94
4.1.3 Reading mesh data to create a STK Mesh, delaying field allocations 95
4.1.4 Outputting STK Mesh 96
4.1.5 Outputting results data from a STK Mesh 96
4.1.6 Outputting a field with an alternative name to a results file 97
4.1.7 Outputting both results and restart data from a STK Mesh 98
4.1.8 Writing multi-state fields to results output file 99
4.1.9 Writing multiple output files 100
4.1.10 Outputting nodal variables on a subset of the nodes 101
4.1.11 Get number of time steps from a database 102
4.1.12 Reading sequenced fields from a database 103
4.1.13 Reading initial conditions from a field on a mesh database 103
4.1.14 Reading initial conditions from a field on a mesh database – apply to a specified subset of mesh parts 104
4.1.15 Reading initial conditions from a field on a mesh database – only read once 107
4.1.16 Reading initial conditions from a mesh database field at a specified database time 109

4.1.17 Reading field data from a mesh database – interpolating between database times . . . 110
4.1.18 Combining restart and interpolation of field data . . . 111
4.1.19 Interpolating field data from a mesh database with only a single database time . . . 112
4.1.20 Interpolating field data from a mesh database when time is outside database time interval . . . 113
4.1.21 Error condition – reading initial conditions from a field that does not exist on a mesh database . . . 115
4.1.22 Interpolation of fields on database with negative times . . . 116
4.1.23 Interpolation of fields on database with non-monotonically increasing times 117
4.1.24 Arbitrary analysis time to database time mapping during field input . . . 118
4.1.25 Error condition – specifying interpolation for an integer field . . . 120
4.1.26 Working with element attributes . . . 121
4.1.27 Create an output mesh with a subset of the mesh parts . . . 122
4.1.28 Writing and reading global variables . . . 123
4.1.29 Writing and reading global parameters . . . 124
4.1.30 Writing global variables automatically . . . 126
4.1.31 Heartbeat output . . . 126
4.1.31.1 Change output precision . . . 128
4.1.31.2 Change field separator . . . 129
4.1.32 Miscellaneous capabilities . . . 129
4.1.32.1 Add contents of a file and/or strings to the information records of a database . . . 129
4.1.32.2 Tell database to overwrite steps instead of adding new steps . . . 130

5 STK Search **133**

5.1 STK Search: usage examples . . . 133

5.1.1 Using Boost R-tree bounding volume search 133
5.1.2 Search method options 134

6 STK Util **135**

6.1 Using the Diagnostic Timers 135
6.2 Communicating with other MPI processors 139
6.3 Using the STK Scheduler 141
6.4 Parameters – type-safe named storage of any variable type 144
6.5 Filename substitution 147

Bibliography **150**

Index **152**

Listings

1.1 Example of creating an STK Mesh using an Exodus file 22

1.2 Example of how to control automatically generated aura 23

1.3 Example of how to enable mesh diagnostics 25

1.4 Example of how to enforce Parallel Mesh Rule 1 26

1.5 Example of communicating field data from owned to all shared and ghosted entities 26

1.6 Example of parallel_sum 27

1.7 Example showing parallel use of comm_mesh_counts 27

1.8 Example showing parallel use of comm_mesh_counts with min/max counts 28

1.9 Example of how to use Selectors to avoid getting caught by the "Nothing" selector 29

1.10 Example of how to use Ghost Parts to select aura ghosts and custom ghosts 33

1.11 Example showing how to use destroy_elements_of_topology 37

1.12 Example showing how to use generate_new_ids 37

1.13 Example showing creation of shared nodes 38

1.14 Example showing creation of independent shared nodes (without connectivity) . . . 39

1.15 Example showing that the marking for independent nodes will be removed after connectivities are attached 40

1.16 Example showing an element being ghosted. 43

1.17 Example of changing processor ownership of an element 44

1.18 Example of internal sideset which results in two faces 45

1.19 Example of how to generate multiple new entities and subsequently set topologies and nodal relations. 48

1.20 Example of how to create all element faces 49

1.21 Example of how to create all element edges and faces 49
1.22 Example of how to create faces on only selected elements 50
1.23 Example showing that faces are created correctly when layered shells are present 51
1.24 Example of how many faces get constructed by CreateFaces between two hexes. . 52
1.25 Example of how many faces get constructed by CreateFaces on a shell. 52
1.26 Example of how many faces get constructed by CreateFaces between hexes and an internal shell. 53
1.27 Example of how to create all the exposed boundary sides 54
1.28 Example of how to create all the interior block boundary sides 55
1.29 Example of how to destroy elements in a list 56
1.30 Example of iterating over nodes 56
1.31 Example of how to traverse connectivity via accessors on BulkData and via accessors on Bucket 58
1.32 Example of how to check side equivalency. 60
1.33 Understanding edge and face ordering 60
1.34 Example showing how to sort entities by descending identifier. 61
2.1 Example of setting/getting topology 64
2.2 Example showing mapping of STK topologies to ranks 64
2.3 Example using compile-time STK topology information 66
2.4 Example showing STK topology for a zero-dimensional element 66
2.5 Example of STK topology for a one-dimensional element 67
2.6 Example of STK topology for a two-dimensional element 68
2.7 Example of STK topology for a three-dimensional element 69
2.8 Example using of an `equivalent` method 71
2.9 Example using `is_positive_polarity` 72
2.10 Example using `lexicographical_smallest_permutation` 72

2.11 Example using `lexicographical_smallest_permutation_preserve_polarity` . 73

2.12 Example using of `sub_topology` 74

2.13 Example for understanding sides in STK topology 75

2.14 Example using a SuperElement with STK topology 75

2.15 Example for understanding various Sierra topologies 76

2.16 Mapping of shards::CellTopologies to stk::topologies provided by stk::mesh::get_cell_topology() 78

3.1 Examples of constant-size whole-mesh field usage 81

3.2 Example of incorrect vector field declaration 82

3.3 Examples of how to get fields by name 83

3.4 Examples of using fields that are variable-size and defined on only a subset of the mesh 83

3.5 Examples of multi-state field usage 84

4.1 Reading mesh data to create a STK mesh 87

4.2 Face creation during IO for one sideset between hexes 89

4.3 Face creation during IO for shells between hexes with sidesets 91

4.4 Reading mesh data to create a STK mesh using set bulk data 94

4.5 Reading mesh data to create a STK mesh; delay field allocation 95

4.6 Writing a STK Mesh 96

4.7 Writing calculated field data to a results database 96

4.8 Outputting a field with an alternative name 97

4.9 Write results and restart 98

4.10 Writing multi-state field to results output 99

4.11 Writing multiple output files 100

4.12 Using a nodeset variable to output nodal fields defined on only a subset of the mesh 101

4.13 get num time steps 102

4.14 Reading sequenced fields 103
4.15 Reading initial condition data from a mesh database 104
4.16 Reading initial condition data from a mesh database 104
4.17 Reading initial condition data from a mesh database 105
4.18 Reading initial condition data from a mesh database 106
4.19 Reading initial condition data from a mesh database one time only 108
4.20 Reading initial condition data from a mesh database at a specified time 109
4.21 Linearly interpolating field data from a mesh database 110
4.22 Combining restart and field interpolation 111
4.23 Linearly interpolating field data from a mesh database with only a single step . . . 113
4.24 Linearly interpolating field data when the time is outside the database time interval 114
4.25 Specifying initial conditions from a non-existent field 115
4.26 Specifying initial conditions from a non-existent field 116
4.27 Interpolating fields on a database with negative times 116
4.28 Interpolating fields on a database with non-monotonically increasing times 117
4.29 Arbitrary analysis time to database time mapping during field input 119
4.30 Error condition – specifying interpolation of an integer field 120
4.31 Working with element attributes 121
4.32 Creating output mesh containing a subset of the mesh parts 122
4.33 Writing and reading a global variable 123
4.34 Writing and reading parameters as global variables 124
4.35 Automatically writing parameters as global variables 126
4.36 Writing global variables to a Heartbeat file 127
4.37 Writing global variables to a Heartbeat file in CSV format with extended precision 128
4.38 Writing global variables to a Heartbeat file with a user-specified field separator . . 129
4.39 Adding the contents of a file to the information records of an output database 129

4.40 Overwriting time steps instead of adding new steps to a database 130

5.1 Using the bounding volume search with the Boost R-tree method 133

5.2 Search method options 134

6.1 Diagnostic Timers 135

6.2 Diagnostic Timers in Parallel 137

6.3 Example showing how to communicate with other processors 139

6.4 Example showing how to communicate an arbitrary amount of data with other processors 140

6.5 Using the scheduler 141

6.6 Parameters: Data for use in the following examples 144

6.7 Parameters: Defining 145

6.8 Parameters: Accessing values 145

6.9 Parameters: Dealing with errors 146

6.10 Parameters: Storing unsupported types 147

6.11 Filename substitution capability 149

This page intentionally left blank.

Chapter 1

STK Mesh

At a high level, the Sierra Toolkit (STK) modules support the engineering science application developer by helping to characterize an unstructured mesh (such as is needed for a Finite Element or Finite Volume mesh) and provide capabilities to support full end-to-end simulations.

Currently, STK is composed of several modules:

- STK Mesh
- STK Search
- STK Transfer*
- STK Rebalance*
- STK IO
- STK Util

*STK Transfer and STK Rebalance will not be discussed in this document.

In the first section of this document, we will introduce STK Mesh terminology and concepts, with the largest effort being towards documenting code usage by using up-to-date examples (that are also in the code repository). This section provides definitions and descriptions of basic STK Mesh terms. Throughout, we use the Exodus [1] mesh format for illustration purposes, and it is recommended that STK Mesh clients be familiar with Exodus.

1.1 STK Mesh Terms

A *Mesh* is a collection of *entities*, *parts*, *fields*, and *field data*. The STK Mesh API separates these collections into *MetaData* and *BulkData*.

Each of these terms is defined below.

1.1.1 Entity

Entity is a general term for the following types (listed in ascending 'rank' order): node, edge, face, element, and constraint. *Rank* is an enumerated type that describes and orders the different kinds of entities.

1.1.2 Connectivity

In a finite element discretization, entities are connected to other entities. Examples include: element-to-node connectivity (the nodes connected to a given element), node-to-element connectivity (the elements connected to a given node), and face-to-element connectivity (the elements connected to a given face). A connection from a higher-rank entity to a lower-rank entity is referred to as a *downward relation*. When a downward relation is declared (e.g., between an element and a node), STK Mesh, by default, creates the corresponding *upward relation* (e.g., from the node to the element). Table 1.1 shows the default connectivity of a fully-connected mesh. The term fully-connected means that the client code has established all downward relations. The term fixed means that the number of relations is defined by topology; the number of node-relations for a hex-8 element is 8. The term dynamic means that the number of relations is unknown until individual relations have been established. For example, an element may have 0, 1, or more faces depending on whether it is on an external boundary. STK mesh provides functions for creating all edges or faces (see Sections 1.6.7 and 1.6.8). It should be noted that STK Mesh does not support connectivity between entities of the same rank. As an additional note, the term *relations* and *connectivity* are used interchangeably in this document.

Table 1.1: Default connectivity of a fully-connected mesh

From-entity	To Node	To Edge	To Face	To Element
Node	-	dynamic	dynamic	dynamic
Edge	fixed	-	dynamic	dynamic
Face	fixed	dynamic	-	dynamic
Element	fixed	dynamic	dynamic	-

1.1.3 Topology

Topology provides an entity's finite element description. This includes several attributes such as the number and type of lower-rank entities that can exist in that entity's downward relations. For example, an element with hex8 topology must have 8 nodes and may have up to a maximum of 6 quad4 faces and 12 line2 edges. Quad4, line2, and nodes are also examples of topologies. Topology also defines what *permutations* in downward connectivity are permissible. Unlike downward connectivity, upward connectivity is determined at run-time and does not imply restrictions on permutations. See chapter 2 for more detail about the STK Topology component and examples of using the API.

Note that in STK Mesh, entities with entity-rank higher than element-rank generally don't have an associated topology.

1.1.4 Part

Part is a general term for a subset of entities in a mesh. Parts are a grouping mechanism used to operate on subsets of the mesh (see Section 1.1.6). STK Mesh automatically creates four parts at startup: the *universal part*, the *locally-owned part*, the *globally-shared part*, and the *aura part*. These parts are important to the basic understanding of ghosting (see Section 1.1.8). For meshes read from Exodus files, additional Exodus parts are created (blocks, sidesets, and nodesets). Each entity in the mesh must be a member of one or more parts.

Parts exist for the life of the STK Mesh; parts cannot be deleted without deleting the mesh. STK Mesh provides methods which allow client code to explicitly change the user-defined part membership of an entity.

See Section 1.5 for more details on mesh parts.

1.1.5 Field

Fields are data associated with mesh entities. Examples include coordinates, velocity, displacement, and temperature. A field in STK Mesh can hold any data type (e.g., double or int) and any number of scalars per entity (e.g., nodal velocity field has three doubles per node if the spatial dimension is 3). A field can be allocated (defined) on the whole mesh (e.g., all nodes) or on a Part (subset) of the mesh (nodes of a sideset). For example, a material property can be defined on a specified element block.

1.1.6 Selector

Selectors are used to select entities that belong to a specified expression of parts. Here are some examples:

- Select all elements that are in either block-1 or block-2 or both. (A set-union expression.)
- Select all nodes that are connected to elements in both block-1 and block-2. (A set-intersection expression.)
- Select all nodes that are locally-owned but not connected to a rigid-body part. (A set-difference expression.)
- Select all nodes that have a specified field allocated. Since field allocation is specified in terms of parts, we allow selectors to be created based on fields.

The selector system is explained further in Section 1.4.

1.1.7 Bucket

STK Mesh organizes entities into *buckets*: the entities in a bucket all have the same rank and topology, and they are all members of the same parts. Additionally, the entities in a bucket correspond to contiguously-allocated blocks of memory in the associated field-data values.

There are two primary reasons for grouping entities into buckets. Firstly, the Selector system (see section 1.4) allows for the traversal of the mesh in arbitrary user-defined subsets, and these subsets exist as combinations of buckets. Secondly, the performance of mesh-modification (see section 1.6) is improved by only moving bucket-sized sections of allocated memory (e.g., when adding/deleting entities) rather than re-allocating and sliding the memory for the whole mesh.

No entity is ever in more than one bucket at any given time. This grouping is performed internally by STK Mesh; client code has no explicit control over which entities reside in which buckets. If an entity's part membership is changed, it is automatically moved to a different bucket.

1.1.8 Ghosting

Ghosting in STK Mesh provides a way to perform operations that involve entities that are neither locally-owned nor shared on the current processor. STK Mesh automatically provides a one-element thick ghost layer around each processor, referred to as the *aura* and is shown in Figures 1.1 and 1.2. Formally, the aura is defined as a ghosting of the upward-relations for shared entities. In other words, if the aura is on, then shared entities have the same upward-relations on each sharing processor. In addition, STK Mesh client code can also request arbitrary ghosting of entities, referred to as custom ghosting.

1.1.9 MetaData and BulkData

The *MetaData* component of a STK Mesh contains the definitions of its parts, the definitions of its fields, and definitions of relationships among its parts and fields. For example, a subset relationship can be declared between two parts, and a field definition can be limited to specific parts. The *BulkData* component of a STK Mesh contains entities, entity ownership and ghosting information, connectivity data, and field data. For efficiency, the BulkData API enables data access via buckets, in addition to data access via entity and rank.

A mesh's MetaData holds a database definition (a schema), and a mesh's BulkData holds the content of that database. MetaData is replicated (duplicated) on all processors; BulkData is distributed across processors with each processor having a separate subset of the data, subject to sharing and ghosting.

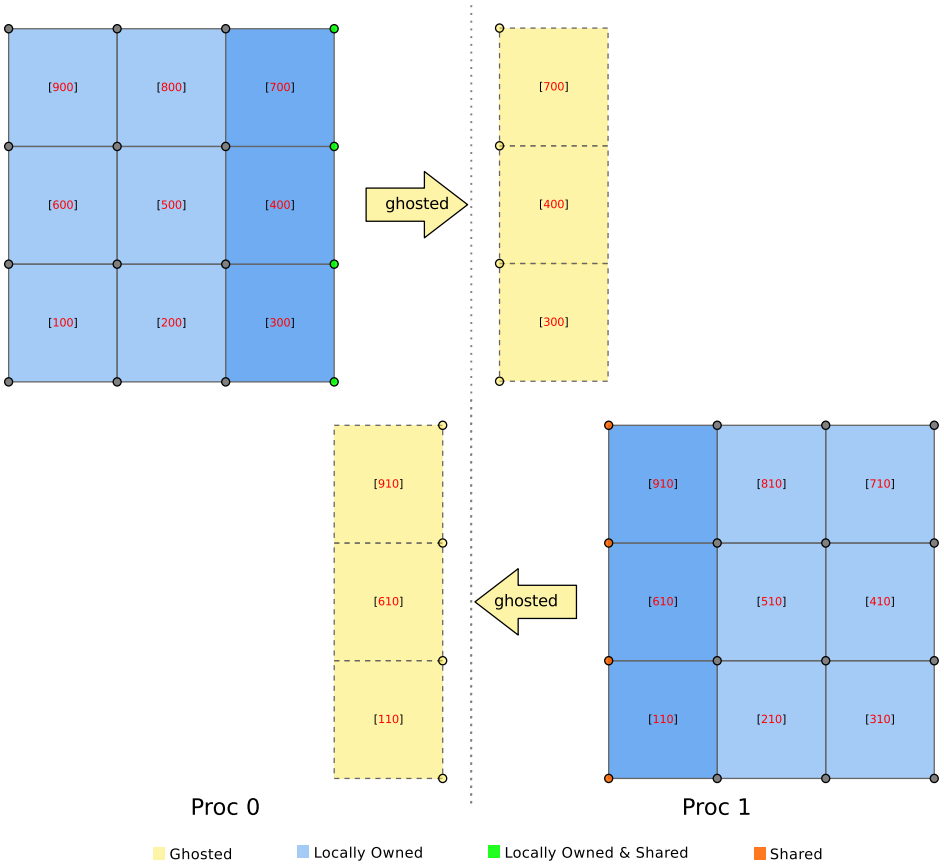

Figure 1.1: Aura ghosting per MPI process

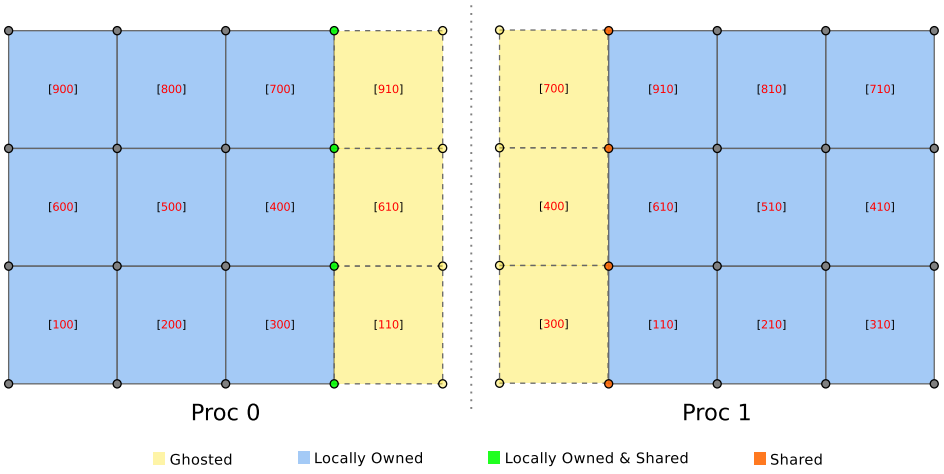

Figure 1.2: Final auras

This design requires object construction of MetaData and BulkData to be staged. The spatial dimension of a mesh is usually specified in the call to the MetaData constructor, which also provides a valid default initialization. The BulkData constructor requires a MetaData object as an argument. A BulkData object cannot be modified (e.g., entities added) before its MetaData object has been initialized and then committed using the `MetaData::commit()` member function

(for example, see Listing 2.1). Once a MetaData object has been committed, it cannot be changed. Therefore, fields must be put on parts prior to MetaData commit. Non-topology parts can still be declared after commit, but they will have limited uses because subset relationships cannot be changed. For clarity, it is recommended that MetaData commit is called prior to BulkData construction. If `new` is used to create a BulkData object, then that instance must be deleted before its MetaData object (used to construct it) is destroyed.

The STK Mesh usage examples below and in Section 1.7 illustrate common uses of the MetaData and BulkData APIs.

1.1.10 Creating a STK Mesh from an Exodus file

Listing 1.1 shows how to create and populate a STK Mesh using the STK IO module, which is described in Chapter 4. We provide this example for those who want to quickly get started using an STK Mesh given an Exodus file. This particular example shows STK IO populating the STK Mesh from a generated-in-memory mesh, but the "filename" is all that would need to change, to instead read data from an Exodus file. Further examples will show various uses of the STK Mesh.

Listing 1.1: Example of creating an STK Mesh using an Exodus file ../../../code/stk/stk_mesh/testsForDocumentation/createStkMesh.cpp

```
49 TEST(StkMeshHowTo, UseStkIO)
50 {
51     MPI_Comm communicator = MPI_COMM_WORLD;
52     if(stk::parallel_machine_size(communicator) == 1)
53     {
54         stk::mesh::MetaData meta;
55         stk::mesh::BulkData bulk(meta, communicator);
56 
57         stk::io::StkMeshIoBroker meshReader;
58         meshReader.set_bulk_data(bulk);
59         meshReader.add_mesh_database("generated:8x8x8", stk::io::READ_MESH);
60         meshReader.create_input_mesh();
61         meshReader.add_all_mesh_fields_as_input_fields();
62         meshReader.populate_bulk_data();
63 
64         unsigned numElems = stk::mesh::count_selected_entities(meta.universal_part(),
               bulk.buckets(stk::topology::ELEM_RANK));
65         EXPECT_EQ(512u, numElems);
66     }
67 }
```

After these steps, the STK Mesh objects now contain all the data from the Exodus file (e.g., Fields, Parts, Entities).

1.2 Parallel

STK Mesh maintains a parallel consistent mesh across many MPI processes or subdomains. Most of the parallel capabilities revolve around communicating information, like field data, for entities

on the boundaries of these subdomains. Entities that are communicated between subdomains are either shared or ghosted.

1.2.1 Shared

Entities that are shared among processors are downward connected from a locally-owned entity, usually an element. For example, if the side of a hex8 is on a subdomain boundary, the 4 nodes that touch the boundary are considered *shared*. If there also exists a face on that side of the hex, the face would also be shared.

Shared entities have fully symmetric communication information stored on all processors that share the entity. In other words, every processor that has a shared entity knows about every other processor that shares the entity.

1.2.2 Ghosted

Ghosted entities are communicated between subdomains regardless of the connections from locally-owned entities. This is different from shared entities which are defined by downward connection from locally-owned entities.

Ghosted entities only have communication information about the owner stored on the processor that the entities are ghosted to. This means that a given processor's BulkData has information about the processor the ghost came from but not any other processors that the entity may have been ghosted to.

1.2.3 Aura

The *aura* is a special ghosting that automatically sends one layer of ghosted elements on the subdomain boundaries to the processors that share those boundaries, as seen in Figures 1.1 and 1.2. The aura can be turned off when the mesh is initially created. See Section 1.2.3.1 for example usage.

1.2.3.1 How to use automatically generated aura

This section describes how to control whether or not a one-layer ghosting of elements is automatically generated around each processor's mesh.

Listing 1.2: Example of how to control automatically generated aura
../../../code/stk/stk_doc_tests/stk_mesh/howToUseAura.cpp

```
48 void expectNumElementsInAura(stk::mesh::BulkData::AutomaticAuraOption autoAuraOption,
```

```
49                              unsigned numExpectedElementsInAura)
50 {
51     MPI_Comm communicator = MPI_COMM_WORLD;
52     if (stk::parallel_machine_size(communicator) == 2)
53     {
54         stk::mesh::MetaData meta;
55         stk::mesh::BulkData bulk(meta, communicator, autoAuraOption);
56         stk::unit_test_util::fill_mesh_using_stk_io("generated:1x1x2", bulk);
57 
58         EXPECT_EQ(numExpectedElementsInAura,
59                   stk::mesh::count_selected_entities(meta.aura_part(),
               bulk.buckets(stk::topology::ELEMENT_RANK)));
60     }
61 }
62 TEST(StkMeshHowTo, useNoAura)
63 {
64     expectNumElementsInAura(stk::mesh::BulkData::NO_AUTO_AURA, 0);
65 }
66 TEST(StkMeshHowTo, useAutomaticGeneratedAura)
67 {
68     expectNumElementsInAura(stk::mesh::BulkData::AUTO_AURA, 1);
69 }
70 TEST(StkMeshHowTo, useAuraDefaultBehavior)
71 {
72     MPI_Comm communicator = MPI_COMM_WORLD;
73     if (stk::parallel_machine_size(communicator) == 2)
74     {
75         stk::mesh::MetaData meta;
76         stk::mesh::BulkData bulk(meta, communicator);
77         stk::unit_test_util::fill_mesh_using_stk_io("generated:1x1x2", bulk);
78 
79         EXPECT_EQ(1u, stk::mesh::count_selected_entities(meta.aura_part(),
               bulk.buckets(stk::topology::ELEMENT_RANK)));
80     }
81 }
```

1.3 STK Parallel Mesh Consistency Rules

STK Mesh is used by many engineering disciplines such as structural dynamics, solid mechanics, thermal/fluid mechanics, and mesh refinement. Since the mesh is being used by different applications, we must ensure that the mesh is **consistent**. A consistent mesh will always follow certain rules/guidelines regardless of the application using it. This has a disadvantage in that flexibility to tune/adjust the mesh for a specific application's needs is reduced, but it also allows easier coupling between applications and helps reuse of algorithms that use STK Mesh because of these rules.

Much of the work in STK Mesh, during modification cycles, is towards creating a consistent mesh especially in parallel. The following are some of the ideas behind a parallel consistent mesh:

- For entities with the same identifier (EntityKey), then for all the processors that have the entity
  - the owner is the same
  - the application-defined parts that the entity is a member of, are the same

  - every entity has the same downward relations on all processors
  - every entity has the same upward relations on all processors (only if the aura is active)

- For aura'ed/shared entities
  - owner of entity knows with which processors the entity is shared with and/or aura'ed to
  - sharer (not owner) of entity knows which other processors share the entity
  - processor with aura'ed entity knows the owner of the entity

At first glance, these rules might seem trivial. The STK Mesh API prevents the ability to change mesh to get it into an inconsistent state at the end of a modification cycle. This concept has proven to be powerful in that it allows coupling of codes and reuse of algorithms across applications.

1.3.1 How to enable mesh diagnostics to enforce parallel mesh rules

STK Mesh now provides a means by which an application may enable internal mesh diagnostics to ensure that the mesh is consistent with the three Parallel Mesh Rules (PMR). These rules may be summarized as:

- Rule 1: Coincident and partially coincident elements must be owned by the same processor (no split coincident elements)
- Rule 2: Each global id shall be owned by one and only one processor (no duplicate ids)
- Rule 3: Processor that owns a side also owns at least one element to which it is connected. (each side needs an element i.e no solo faces)

Enabling mesh diagnostics creates a per-processor file named "mesh_diagnostics_failures_<proc_id>.txt" which contains the listing of all errors. This example demonstrates first creating a mesh with a sideset and then checking that there are no solo faces with attached elements that are remotely owned (PMR-3).

Listing 1.3: Example of how to enable mesh diagnostics
../../../code/stk/stk_mesh/testsForDocumentation/howToEnableMeshDiagnostics.cpp

```
43 TEST(StkMeshHowTo, EnableMeshDiagnostics)
44 {
45     stk::mesh::MetaData meta;
46     stk::mesh::BulkData bulkData(meta, MPI_COMM_WORLD);
47     stk::unit_test_util::fill_mesh_using_stk_io("generated:4x4x4|sideset:xX", bulkData);
48
49     bulkData.enable_mesh_diagnostic_rule(stk::mesh::RULE_3);
50     EXPECT_EQ(0u, bulkData.get_mesh_diagnostic_error_count());
51 }
```

1.3.2 How to enforce Parallel Mesh Rule 1

STK Mesh now provides a means by which an application may enforce Parallel Mesh Rule 1 (PMR-1) to ensure that coincident and partially-coincident elements must be owned by the same processor (no split coincident elements).

Listing 1.4: Example of how to enforce Parallel Mesh Rule 1
../../../code/components/penso/doc_tests/howToFixPMR1Violation.cpp

```
43 TEST(StkMeshHowTo, FixPMR1Violation)
44 {
45     stk::mesh::MetaData meta;
46     stk::mesh::BulkData bulkData(meta, MPI_COMM_WORLD);
47     stk::unit_test_util::fill_mesh_using_stk_io("generated:4x4x4|sideset:xX", bulkData);
48
49     stk::mesh::EntityIdProcMap elementAndDestProc;
50     EXPECT_NO_THROW(elementAndDestProc =
               penso::make_mesh_consistent_with_parallel_mesh_rule1(bulkData));
51     EXPECT_TRUE(elementAndDestProc.size()==0u); // no elements were migrated
52 }
```

1.3.3 Parallel API

This section discusses a few API functions for applications using the parallel capabilities of STK Mesh.

The following code example shows how to communicate field data from owned to all shared and ghosted entities, overwriting any local modifications.

Listing 1.5: Example of communicating field data from owned to all shared and ghosted entities
../../../code/stk/stk_mesh/testsForDocumentation/communicateFieldData.cpp

```
59 TEST_F(ParallelHowTo, communicateFieldDataForSharedAndAura)
60 {
61     auto& field =
           get_meta().declare_field<stk::mesh::Field<double>>(stk::topology::NODE_RANK,
           "temperature");
62
63     double initialValue = 25.0;
64     stk::mesh::put_field_on_entire_mesh_with_initial_value(field, &initialValue);
65
66     setup_mesh("generated:8x8x8", stk::mesh::BulkData::AUTO_AURA);
67
68     const stk::mesh::BucketVector& notOwnedBuckets =
           get_bulk().get_buckets(stk::topology::NODE_RANK,
69
           !get_meta().locally_owned_part());
70
71     for(const stk::mesh::Bucket *bucket : notOwnedBuckets)
72         for(stk::mesh::Entity node : *bucket)
73             *stk::mesh::field_data(field, node) = -1.2345;
74
75     stk::mesh::communicate_field_data(get_bulk(), {&field});
76
77     for(const stk::mesh::Bucket *bucket : notOwnedBuckets)
78         for(stk::mesh::Entity node : *bucket)
79             EXPECT_EQ(initialValue, *stk::mesh::field_data(field, node));
```

```
80 }
```

The parallel_sum, parallel_min, and parallel_max functions operate on shared entities.

Listing 1.6: Example of parallel_sum
../../../code/stk/stk_mesh/testsForDocumentation/communicateFieldData.cpp

```
84 void expect_field_has_value(const stk::mesh::BucketVector& buckets,
85                              const stk::mesh::Field<double> &field,
86                              double value)
87 {
88     for(const stk::mesh::Bucket *bucket : buckets)
89         for(stk::mesh::Entity node : *bucket)
90             EXPECT_EQ(value, *stk::mesh::field_data(field, node));
91 }
92
93 TEST_F(ParallelHowTo, computeParallelSum)
94 {
95     auto& field =
              get_meta().declare_field<stk::mesh::Field<double>>(stk::topology::NODE_RANK,
              "temperature");
96
97     double initialValue = 25.0;
98     stk::mesh::put_field_on_entire_mesh_with_initial_value(field, &initialValue);
99
100    setup_mesh("generated:8x8x8", stk::mesh::BulkData::AUTO_AURA);
101
102    const stk::mesh::BucketVector& shared = get_bulk().get_buckets(stk::topology::NODE_RANK,
103
              get_meta().globally_shared_part());
104    const stk::mesh::BucketVector& notShared =
              get_bulk().get_buckets(stk::topology::NODE_RANK,
105
              !get_meta().globally_shared_part());
106    expect_field_has_value(shared, field, initialValue);
107    expect_field_has_value(notShared, field, initialValue);
108
109    stk::mesh::parallel_sum(get_bulk(), {&field});
110
111    expect_field_has_value(shared, field, 2*initialValue);
112    expect_field_has_value(notShared, field, initialValue);
113 }
```

The `comm_mesh_counts` function is shown in Listings 1.7-1.8. The purpose of this function is to count the number of entities of each entity rank across all processors.

Listing 1.7: Example showing parallel use of comm_mesh_counts
../../../code/stk/stk_doc_tests/stk_mesh/UnitTestCommMeshCounts.cpp

```
75 TEST( CommMeshCounts, Parallel )
76 {
77     stk::ParallelMachine communicator = MPI_COMM_WORLD;
78     int numprocs = stk::parallel_machine_size(communicator);
79
80     const std::string generatedMeshSpec = getGeneratedMeshString(10,20,2*numprocs);
81     unitTestUtils::exampleMeshes::StkMeshCreator stkMesh(generatedMeshSpec, communicator);
82
83     std::vector<size_t> comm_mesh_counts;
84     stk::mesh::comm_mesh_counts(*stkMesh.getBulkData(), comm_mesh_counts);
85
86     size_t goldNumElements = 10*20*2*numprocs;
87     EXPECT_EQ(goldNumElements, comm_mesh_counts[stk::topology::ELEMENT_RANK]);
```

```
88 }
```

Listing 1.8: Example showing parallel use of comm_mesh_counts with min/max counts ../../../code/stk/stk_doc_tests/stk_mesh/UnitTestCommMeshCounts.cpp

```
90 TEST( CommMeshCountsWithStats, Parallel )
91 {
92     stk::ParallelMachine communicator = MPI_COMM_WORLD;
93     int numprocs = stk::parallel_machine_size(communicator);
94
95     const std::string generatedMeshSpec = getGeneratedMeshString(10,20,2*numprocs);
96     unitTestUtils::exampleMeshes::StkMeshCreator stkMesh(generatedMeshSpec, communicator);
97
98     std::vector<size_t> comm_mesh_counts;
99     std::vector<size_t> min_counts;
100    std::vector<size_t> max_counts;
101
102    stk::mesh::comm_mesh_counts(*stkMesh.getBulkData(), comm_mesh_counts, min_counts,
               max_counts);
103
104    size_t goldNumElements = 10*20*2*numprocs;
105    EXPECT_EQ(goldNumElements, comm_mesh_counts[stk::topology::ELEMENT_RANK]);
106
107    size_t goldMinNumElements = 10*20*2;
108    EXPECT_EQ(goldMinNumElements, min_counts[stk::topology::ELEMENT_RANK]);
109
110    size_t goldMaxNumElements = goldMinNumElements;
111    EXPECT_EQ(goldMaxNumElements, max_counts[stk::topology::ELEMENT_RANK]);
112 }
```

1.4 STK Mesh Selector

A *selector* is a set-logical expression that can include intersections, unions, and complements. The default-constructed selector is empty and therefore selects nothing. See Section 1.4.1 for examples.

A selector is typically used with `get_buckets()` for a given entity rank to get a list of buckets satisfying that selector. `get_buckets()` evaluates the selector on each bucket of the specified rank. When the expression evaluation gets down to a part, the selector must determine if that part is listed as one of the part intersections in the bucket. The worst-case cost of evaluating `get_buckets()` is

$$O\left(N_{number\ buckets}\right) \times O\left(N_{number\ selector\ terms}\right) \times O\left(N_{number\ bucket\ parts}\right) \tag{1.1}$$

where $N_{number\ buckets}$ is the number of buckets of the Entity rank that was passed into `get_buckets()`, $N_{number\ selector\ terms}$ is the length of the selector expression, and $N_{number\ bucket\ parts}$ is the average number of parts that each bucket represents.

Since STK Mesh internally caches the results of calls to `get_buckets()`, selector performance often does not have a large impact on overall application runtime. Selectors are implemented to allow optimization from short-circuiting logic, to allow a positive result from a union to ignore the rest of the expression, as well as a negative result from an intersection. If selectors are constructed

to take advantage of this type of early termination, the middle term in equation (1.1) is less expensive in practice. For example, if `partA` strictly contains `partB`, then the selector expression `(partA | partB)` will tend to select more efficiently than `(partB | partA)` because, in the first case, once it is known that a bucket is selected for `partA`, that bucket does not need to be checked against `partB`.

1.4.1 How to use selectors

These examples demonstrate creating and printing Selectors, as well as performing set intersection operations. The second example also demonstrates retrieving the buckets associated with a Selector.

Listing 1.9: Example of how to use Selectors to avoid getting caught by the "Nothing" selector ../../../code/stk/stk_mesh/testsForDocumentation/howToUseSelectors.cpp

```
50 TEST(StkMeshHowTo, betterUnderstandSelectorConstruction)
51 {
52     MPI_Comm communicator = MPI_COMM_WORLD;
53     if (stk::parallel_machine_size(communicator) != 1) { return; }
54     stk::io::StkMeshIoBroker stkMeshIoBroker(communicator);
55     const std::string generatedMeshSpecification = "generated:1x1x1"; // syntax creates a
           1x1x1 cube
56     stkMeshIoBroker.add_mesh_database(generatedMeshSpecification, stk::io::READ_MESH);
57     stkMeshIoBroker.create_input_mesh();
58     stkMeshIoBroker.populate_bulk_data();
59
60     stk::mesh::BulkData &stkMeshBulkData = stkMeshIoBroker.bulk_data();
61
62     stk::mesh::Selector nothingSelector_byDefaultConstruction;
63     size_t expectingZeroBuckets = 0;
64     EXPECT_EQ(expectingZeroBuckets, stkMeshBulkData.get_buckets(stk::topology::NODE_RANK,
           nothingSelector_byDefaultConstruction).size());
65
66     std::ostringstream readableSelectorDescription;
67     readableSelectorDescription << nothingSelector_byDefaultConstruction;
68     EXPECT_STREQ("NOTHING", readableSelectorDescription.str().c_str());
69
70     stk::mesh::Selector allSelector(!nothingSelector_byDefaultConstruction);
71     size_t numberOfAllNodeBuckets = stkMeshBulkData.buckets(stk::topology::NODE_RANK).size();
72     EXPECT_EQ(numberOfAllNodeBuckets, stkMeshBulkData.get_buckets(stk::topology::NODE_RANK,
           allSelector).size());
73 }
74
75 TEST(StkMeshHowTo, makeSureYouAreNotIntersectingNothingSelector)
76 {
77     MPI_Comm communicator = MPI_COMM_WORLD;
78     if (stk::parallel_machine_size(communicator) != 1) { return; }
79     stk::io::StkMeshIoBroker stkMeshIoBroker(communicator);
80     // syntax creates faces for surface on the positive: 'x-side', 'y-side', and 'z-side'
81     // of a 1x1x1 cube, these parts are given the names: 'surface_1', 'surface_2', and
           'surface_3'
82     // automagically when it is created [create_input_mesh()]
83     const std::string generatedMeshSpecification = "generated:1x1x1|sideset:XYZ";
84     stkMeshIoBroker.add_mesh_database(generatedMeshSpecification, stk::io::READ_MESH);
85     stkMeshIoBroker.create_input_mesh();
86     stkMeshIoBroker.populate_bulk_data();
87
88     stk::mesh::MetaData &stkMeshMetaData = stkMeshIoBroker.meta_data();
89     stk::mesh::Part *surface1Part = stkMeshMetaData.get_part("surface_1");
90     stk::mesh::Part *surface2Part = stkMeshMetaData.get_part("surface_2");
```

```
91      stk::mesh::Part *surface3Part = stkMeshMetaData.get_part("surface_3");
92      stk::mesh::PartVector allSurfaces;
93      allSurfaces.push_back(surface1Part);
94      allSurfaces.push_back(surface2Part);
95      allSurfaces.push_back(surface3Part);
96
97      stk::mesh::Selector selectorIntersectingNothing;
98      for (size_t surfaceIndex = 0; surfaceIndex < allSurfaces.size(); ++surfaceIndex)
99      {
100         stk::mesh::Part &surfacePart = *(allSurfaces[surfaceIndex]);
101         stk::mesh::Selector surfaceSelector(surfacePart);
102         selectorIntersectingNothing &= surfacePart;
103     }
104     size_t expectedNumberOfBucketsWhenIntersectingNothing = 0;
105     stk::mesh::BulkData &stkMeshBulkData = stkMeshIoBroker.bulk_data();
106     stk::mesh::BucketVector selectedBuckets =
                stkMeshBulkData.get_buckets(stk::topology::NODE_RANK,
                selectorIntersectingNothing);
107     EXPECT_EQ(expectedNumberOfBucketsWhenIntersectingNothing, selectedBuckets.size());
108
109     stk::mesh::Selector preferredBoundaryNodesSelector =
                stk::mesh::selectIntersection(allSurfaces);
110     size_t expectedNumberOfNodeBucketsWhenIntersectingAllSurfaces = 1;
111     selectedBuckets = stkMeshBulkData.get_buckets(stk::topology::NODE_RANK,
                preferredBoundaryNodesSelector);
112     EXPECT_EQ(expectedNumberOfNodeBucketsWhenIntersectingAllSurfaces, selectedBuckets.size());
113 }
```

1.5 STK Mesh Parts

A *mesh part* is a subset of entities of the mesh, and may be used to reflect the physics modeled, discretization methodology, solution algorithm, meshing artifacts, or other application specific requirements.

STK Mesh automatically defines several parts during initialization, demonstrated here based on the serial Exodus mesh and its parallel decomposition previously shown in Figures **??** and **??**, respectively. The *universal part* includes every entity on the current MPI process (Figure 1.3). The *locally-owned part* contains all the entities owned by the current MPI process (Figure 1.4). The *globally-shared part* contains all the entities on the current MPI process that are shared with another MPI process, whether locally-owned or not. Figures 1.5 and 1.6 illustrate the globally shared part. An entity may be in both the locally-owned and globally-shared parts. By default, a shared entity is owned by the lowest-numbered sharing MPI process, though client code is allowed to change entity ownership. Part declarations and part membership are consistent across processor ranks; part membership for a given entity is maintained on the owning rank. The *aura part* contains all the entities which are ghosted due to aura. An additional part is kept up-to-date for each custom ghosting and examples of usage are in Section 1.5.3.

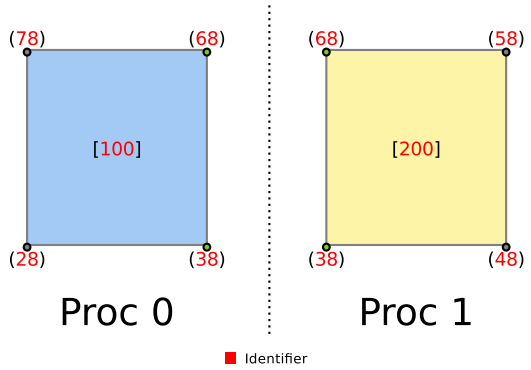

Figure 1.3: Parallel-decomposed STK Mesh. This figure depicts the universal parts on each process.

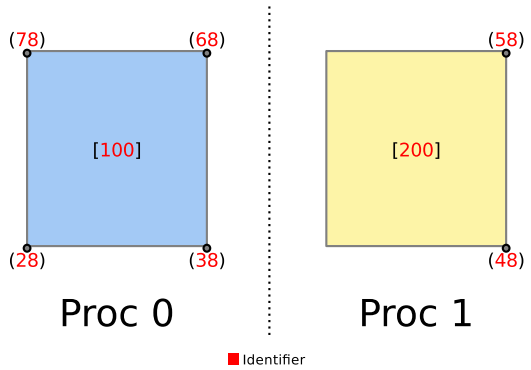

Figure 1.4: Locally-owned parts. Nodes 38 and 68 are owned by process 1 and are not in process 2's locally-owned part.

1.5.1 Part Identifiers and Attributes

A mesh part has an unique text name identifier, specified by the application that creates the part. This identifier is intended to support text input and output by the application, e.g., parsing, logging, and error reporting. The text name is not intended for referencing a mesh part within application computations. As reliance on text-based references will lead to text-based searches within the application's computations, resulting in unnecessarily degraded performance.

A mesh part also has a unique non-negative integer identifier, its *part ordinal*, that is internally generated by the mesh MetaData. Part ordinals are intended to support fast referencing and ordering of mesh parts. The part ordinal is also intended to support efficient communication of mesh part information among distributed memory processes.

An application, for example, may specify a mesh part for an *element block* (a collection of elements); in descriptions of part behavior, we use the following notation:

$$\begin{aligned} Part_A &\equiv \text{mesh part identified by } A \\ Part_A^J &\equiv \text{mesh part intended for mesh entities of rank } J \text{ and identified by } A \end{aligned} \tag{1.2}$$

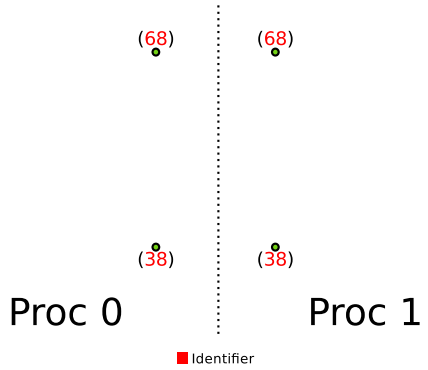

Figure 1.5: Globally-shared parts. Nodes 38 and 68 appear in both process's globally-shared part.

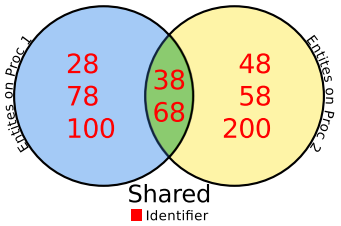

Figure 1.6: Entities in the globally-shared part from each process.

Note that all processors have the same part list. Hence, parts must be created synchronously across all processors to avoid part lists becoming different on any processor.

1.5.2 Induced Part Membership

An application can explicitly insert a mesh entity into a mesh part or explicitly remove a mesh entity from a part. A mesh entity's membership in a part may also be induced through its connectivity to a higher rank mesh entity. Thus, a mesh entity may be an *explicit member* or an *induced member* of a mesh part.

For example, a node will have induced membership in an element block (mesh part) when that node has connectivity from an element that is in that part. Therefore, the nodes of all the elements in the element block will be in that part due to induced part membership. This enables client code to select and iterate over the nodes of the elements in the element block directly and uniquely, rather than through element connectivity. In general, the explicit part membership of a given entity automatically induces the same part membership onto any lower-ranking entities that are connected to it.

When a mesh part has a specified entity rank ($Part_A^J$) then only mesh entities of the same entity rank J may be explicitly added as members to that mesh part. If a mesh entity is an *explicit member* of such a mesh part, $entity_a^J \in Part_A^J$, and that mesh entity ($entity_a^J$) is the from-entity of a connectivity,

then the to-entity of that connectivity is an *induced member* of that mesh part. More formally,

$$\begin{aligned} \text{Given} \quad & \text{a connectivity } \left(entity_a^J \, , \, entity_b^K \, , \, x \right) \, : \, J > K \quad \text{and} \\ & entity_a^J \in Part_A^J \quad \text{via explicit membership} \\ \text{then} \quad & entity_b^K \in Part_A^J \quad \text{via induced membership.} \end{aligned} \tag{1.3}$$

Note that induced-part memberships are added (or removed) whenever a connectivity is declared (or deleted). As a result, declaring or deleting a connectivity can cause an entity to move to a different bucket.

Induced membership only occurs in the presence of a mesh entity connectivity. This means that induced membership is **not** transitive. For example, if a mesh has both element-to-face and face-to-edge connectivities, but does not have element-to-edge connectivities, then the edges in the element's closure (via element-to-face-to-edge) are **not** induced members.

1.5.3 How to use ghost parts

These examples demonstrate how to use the ghost parts to select those entities that are ghosted due to aura or custom ghosting.

Listing 1.10: Example of how to use Ghost Parts to select aura ghosts and custom ghosts ../../../code/stk/stk_doc_tests/stk_mesh/UnitTestGhostParts.cpp

```
66 TEST(UnitTestGhostParts, Aura)
67 {
68   stk::ParallelMachine communicator = MPI_COMM_WORLD;
69
70   int numProcs = stk::parallel_machine_size(communicator);
71   if (numProcs != 2) {
72     return;
73   }
74
75   stk::io::StkMeshIoBroker stkMeshIoBroker(communicator);
76   const std::string generatedMeshSpecification = "generated:1x1x3";
77   stkMeshIoBroker.add_mesh_database(generatedMeshSpecification, stk::io::READ_MESH);
78   stkMeshIoBroker.create_input_mesh();
79   stkMeshIoBroker.populate_bulk_data();
80
81   stk::mesh::MetaData &stkMeshMetaData = stkMeshIoBroker.meta_data();
82   stk::mesh::BulkData &stkMeshBulkData = stkMeshIoBroker.bulk_data();
83
84   std::cerr<<"about to get aura_part..."<<std::endl;
85   stk::mesh::Part& aura_part = stkMeshMetaData.aura_part();
86   std::cerr<<"...got aura part with name="<<aura_part.name()<<std::endl;
87   stk::mesh::Selector aura_selector = aura_part;
88
89   stk::mesh::Ghosting& aura_ghosting = stkMeshBulkData.aura_ghosting();
90   EXPECT_EQ(aura_part.mesh_meta_data_ordinal(),
             stkMeshBulkData.ghosting_part(aura_ghosting).mesh_meta_data_ordinal());
91
92   stk::mesh::Selector not_owned_nor_shared = (!stkMeshMetaData.locally_owned_part()) &
             (!stkMeshMetaData.globally_shared_part());
93
94   const stk::mesh::BucketVector& not_owned_nor_shared_node_buckets =
             stkMeshBulkData.get_buckets(stk::topology::NODE_RANK, not_owned_nor_shared);
```

```
95   size_t expected_num_not_owned_nor_shared_node_buckets = 1;
96   EXPECT_EQ(expected_num_not_owned_nor_shared_node_buckets,
               not_owned_nor_shared_node_buckets.size());
97
98   const stk::mesh::BucketVector& aura_node_buckets =
               stkMeshBulkData.get_buckets(stk::topology::NODE_RANK, aura_selector);
99
100  EXPECT_EQ(not_owned_nor_shared_node_buckets.size(), aura_node_buckets.size());
101
102  const size_t expected_num_ghost_nodes = 4;
103  size_t counted_nodes = 0;
104  size_t counted_aura_nodes = 0;
105  for(size_t i=0; i<not_owned_nor_shared_node_buckets.size(); ++i)
106  {
107      counted_nodes += not_owned_nor_shared_node_buckets[i]->size();
108      counted_aura_nodes += aura_node_buckets[i]->size();
109  }
110  EXPECT_EQ(expected_num_ghost_nodes, counted_nodes);
111  EXPECT_EQ(expected_num_ghost_nodes, counted_aura_nodes);
112 }
113
114 TEST(UnitTestGhostParts, Custom1)
115 {
116   stk::ParallelMachine communicator = MPI_COMM_WORLD;
117
118   int numProcs = stk::parallel_machine_size(communicator);
119   if (numProcs != 2) {
120     return;
121   }
122
123   stk::io::StkMeshIoBroker stkMeshIoBroker(communicator);
124   const std::string generatedMeshSpecification = "generated:1x1x4";
125   stkMeshIoBroker.add_mesh_database(generatedMeshSpecification, stk::io::READ_MESH);
126   stkMeshIoBroker.create_input_mesh();
127   stkMeshIoBroker.populate_bulk_data();
128
129   stk::mesh::BulkData &stkMeshBulkData = stkMeshIoBroker.bulk_data();
130
131   int myProc = stkMeshBulkData.parallel_rank();
132   int otherProc = (myProc == 0) ? 1 : 0;
133
134   stkMeshBulkData.modification_begin();
135
136   stk::mesh::Ghosting& custom_ghosting = stkMeshBulkData.create_ghosting("CustomGhosting1");
137
138   std::vector<stk::mesh::EntityProc> elems_to_ghost;
139
140   const stk::mesh::BucketVector& elem_buckets =
               stkMeshBulkData.buckets(stk::topology::ELEM_RANK);
141   for(size_t i=0; i<elem_buckets.size(); ++i) {
142     const stk::mesh::Bucket& bucket = *elem_buckets[i];
143     for(size_t j=0; j<bucket.size(); ++j) {
144       if (stkMeshBulkData.parallel_owner_rank(bucket[j]) == myProc) {
145         elems_to_ghost.push_back(std::make_pair(bucket[j], otherProc));
146       }
147     }
148   }
149
150   stkMeshBulkData.change_ghosting(custom_ghosting, elems_to_ghost);
151
152   stkMeshBulkData.modification_end();
153
154   //now each processor should have 2 elements that were received as ghosts of elements from
               the other proc.
155   const size_t expected_num_elems_for_custom_ghosting = 2;
156
157   stk::mesh::Part& custom_ghost_part = stkMeshBulkData.ghosting_part(custom_ghosting);
158   stk::mesh::Selector custom_ghost_selector = custom_ghost_part;
```

```
159
160   const stk::mesh::BucketVector& custom_ghost_elem_buckets =
                stkMeshBulkData.get_buckets(stk::topology::ELEM_RANK, custom_ghost_selector);
161   size_t counted_elements = 0;
162   for(size_t i=0; i<custom_ghost_elem_buckets.size(); ++i) {
163     counted_elements += custom_ghost_elem_buckets[i]->size();
164   }
165
166   EXPECT_EQ(expected_num_elems_for_custom_ghosting, counted_elements);
167 }
```

1.6 Mesh Modification

1.6.1 Overview

The following types of mesh modifications are available in STK Mesh:

- Add/delete entities
- Change entities' part membership
- Change connectivity
- Change processors' entity ownership
- Change ghosting

A STK Mesh can be modified only within the context of a *modification cycle*. A modification cycle begins with a call to `BulkData::modification_begin()` and ends when the next call to `BulkData::modification_end()` returns. This latter function does a pre-determined set of checks on mesh status and performs MPI communication to ensure a globally-consistent state.

Modification cycles should not be nested; `BulkData::modification_end()` terminates all "enclosing" modification cycles. If the application inadvertently nests modification cycles, errors are likely to be thrown.

Application code between a `BulkData::modification_begin()` call and the following `BulkData::modification_end()` call can use STK Mesh modification functions that cause the BulkData to become parallel inconsistent. That is, mesh information on different processor ranks can disagree. After each modification cycle, a STK mesh is guaranteed to be parallel-consistent. Failures during mesh modification are not recoverable.

The first time `BulkData::modification_begin()` is called, the mesh MetaData is verified to have been committed and to be parallel-consistent (and the MetaData is committed at that time if it hasn't already been committed). The function returns **`true`** if the mesh successfully transitions from the guaranteed parallel-consistent state to the *MODIFIABLE* state, and **`false`** if it is already in this state.

`BulkData::modification_end()` performs parallel synchronization of local mesh modifications since the mesh entered the *MODIFIABLE* state and transitions the mesh back to a guaranteed parallel-consistent state. `BulkData::modification_end()` returns **`true`** if it succeeds and **`false`** if it is already in the guaranteed parallel-consistent state. If modification resolution errors occur then a parallel-consistent exception will be thrown.

Because a modification cycle incurs multiple rounds of communication and traversal over large portions of the mesh, even a modification cycle with a single modification incurs significant cost. From a performance standpoint it is advantageous to group mesh modifications into as few modification cycles as possible.

To alleviate the expense of a general modification cycle, other single-purpose API have been introduced, such as for the creation of faces, that take into account knowledge of what has been modified to improve the performance of a modification cycle. These should be considered before coding a general modification, especially if it is in a performance-critical part of the code.

Note that *MetaData* changes (declaring parts and fields) are not part of the mesh modification API since it's illegal to change MetaData after the MetaData object has been committed.

1.6.2 Public Modification Capability

In this section we describe the modification operations intended to be called from application code. As noted above, these functions can only be called between calls to `BulkData::modification_begin()` and `BulkData::modification_end()`. We also describe the modification operations that STK Mesh automatically performs internally as a result of an application explicitly calling a modification function. Understanding what modifications can occur automatically is particularly important for code reliability. We note that certain modification types are applicable only in distributed STK Mesh applications.

1.6.2.1 Add/Delete Entities

The `BulkData::declare_entity()` function can be used to add an entity to a STK mesh and assign its entity rank and global identifier. `BulkData::generate_new_entities()` can be used to create multiple entities of specified entity ranks and have unique global identifiers automatically assigned. When entities of EDGE_RANK, FACE_RANK, or ELEMENT_RANK are created by application code, they must be assigned a topology and have their nodal connectivities set before `BulkData::modification_end()` is called. See section 1.6.6.

`BulkData::destroy_entity()` deletes an entity from a STK Mesh. All upward relations must be deleted before an entity can be destroyed, as a safety measure to ensure that the user is explicitly aware of any possible inconsistent mesh states that they are creating (e.g. an element that is missing one or more nodes). Downward relations are deleted automatically.

Adding or deleting an entity can result in automatic changes to part membership, ownership, connectivity, ghosting, and sharing. Changes in part membership(s) can also result in changes to bucket structure. Any local modifications to an entity will cause ghosted copies of that entity to be deleted from other processor ranks. The ghosts will be automatically regenerated if they are part of the aura.

Unless an entity is deleted, it stays valid before, during, and after a modification cycle.

Listing 1.11: Example showing optimized destruction of all elements of a specified topology ../../../code/stk/stk_doc_tests/stk_mesh/howToDestroyElementsOfTopology.cpp

```
1  #include <gtest/gtest.h>
2  #include <stk_mesh/base/BulkData.hpp>
3  #include <stk_mesh/base/GetEntities.hpp>
4  #include <stk_mesh/base/MetaData.hpp>
5  #include <stk_topology/topology.hpp>
6  #include <stk_unit_test_utils/ioUtils.hpp>
7  namespace
8  {
9  TEST(StkMeshHowTo, DestroyElementsOfTopology)
10 {
11     stk::mesh::MetaData metaData;
12     stk::mesh::BulkData bulkData(metaData, MPI_COMM_WORLD);
13     stk::unit_test_util::fill_mesh_using_stk_io("generated:1x1x4", bulkData);
14     EXPECT_GT(stk::mesh::count_selected_entities(metaData.universal_part(),
               bulkData.buckets(stk::topology::ELEM_RANK)), 0u);
15     bulkData.destroy_elements_of_topology(stk::topology::HEX_8);
16     EXPECT_EQ(0u, stk::mesh::count_selected_entities(metaData.universal_part(),
               bulkData.buckets(stk::topology::ELEM_RANK)));
17 }
18 }
```

1.6.2.2 Getting Unused Globally Unique Identifiers

Code Listing 1.12 shows, by example, how to get globally unique identifiers. The API requires that a stk topology rank be specified. The ids are then returned in the vector argument. These ids are unused when this call is made. Hence, care must be taken if these ids are kept on the application side (client side) and not used until later. This is a collective call (all processors must call this function). Note, this API is offered in addition to the `generate_new_entities()` method. The key difference is that the `generate_new_ids()` method only obtains identifiers per rank, and entities are not automatically created.

Listing 1.12: Example showing how to use generate_new_ids ../../../code/stk/stk_doc_tests/stk_mesh/howToUseGenerateNewIds.cpp

```
76 TEST(StkMeshHowTo, use_generate_new_ids)
77 {
78     MPI_Comm communicator = MPI_COMM_WORLD;
79
80     int num_procs = -1;
81     MPI_Comm_size(communicator, &num_procs);
82     std::ostringstream os;
83     os << "generated:1x1x" << num_procs;
84     const std::string generatedMeshSpecification = os.str();
85
86     stk::io::StkMeshIoBroker stkMeshIoBroker(communicator);
```

```
87      stkMeshIoBroker.add_mesh_database(generatedMeshSpecification, stk::io::READ_MESH);
88      stkMeshIoBroker.create_input_mesh();
89      stkMeshIoBroker.populate_bulk_data();
90
91      stk::mesh::BulkData &stkMeshBulkData = stkMeshIoBroker.bulk_data();
92
93      // Given a mesh, request 10 unique node ids
94
95      std::vector<stk::mesh::EntityId> requestedIds;
96      unsigned numRequested = 10;
97
98      stkMeshBulkData.generate_new_ids(stk::topology::NODE_RANK, numRequested, requestedIds);
99
100     test_that_ids_are_unique(stkMeshBulkData, stk::topology::NODE_RANK, requestedIds);
101 }
```

1.6.2.3 Creating Nodes that are Shared by Multiple Processors

When a node entity is created that is intended to be shared by multiple processors (i.e., it will be connected to locally-owned entities on multiple MPI processors), the method `BulkData::add_node_sharing()` must be used to inform STK Mesh that the node is shared and which other processors share it. The `add_node_sharing()` method must be called symmetrically, meaning that for a given shared node, each sharing processor must inform STK Mesh about all the other sharing processors during the same modification cycle. The code listing 1.13 demonstrates the use of `add_node_sharing()` when creating shared nodes.

Listing 1.13: Example showing creation of shared nodes
../../../code/stk/stk_doc_tests/stk_mesh/createSharedNodes.cpp

```
73  TEST(stkMeshHowTo, createSharedNodes)
74  {
75      const unsigned spatialDimension = 2;
76      stk::mesh::MetaData metaData(spatialDimension, stk::mesh::entity_rank_names());
77      stk::mesh::Part &triPart = metaData.declare_part_with_topology("tri_part",
                stk::topology::TRIANGLE_3_2D);
78      metaData.commit();
79
80      stk::mesh::BulkData bulkData(metaData, MPI_COMM_WORLD);
81      if (bulkData.parallel_size() == 2)
82      {
83          bulkData.modification_begin();
84
85          const unsigned nodesPerElem = 3;
86          stk::mesh::EntityIdVector elemIds = {1, 2};//one elemId for each proc
87          std::vector<stk::mesh::EntityIdVector> elemNodeIds = { {1, 3, 2}, {4, 2, 3} };
88          const int myproc = bulkData.parallel_rank();
89
90          stk::mesh::Entity elem = bulkData.declare_entity(stk::topology::ELEM_RANK,
                elemIds[myproc], triPart);
91          stk::mesh::EntityVector elemNodes(nodesPerElem);
92          elemNodes[0] = bulkData.declare_entity(stk::topology::NODE_RANK,
                elemNodeIds[myproc][0]);
93          elemNodes[1] = bulkData.declare_entity(stk::topology::NODE_RANK,
                elemNodeIds[myproc][1]);
94          elemNodes[2] = bulkData.declare_entity(stk::topology::NODE_RANK,
                elemNodeIds[myproc][2]);
95
96          bulkData.declare_relation(elem, elemNodes[0], 0);
97          bulkData.declare_relation(elem, elemNodes[1], 1);
```

```
98          bulkData.declare_relation(elem, elemNodes[2], 2);
99
100         int otherproc = testUtils::get_other_proc(myproc);
101         bulkData.add_node_sharing(elemNodes[1], otherproc);
102         bulkData.add_node_sharing(elemNodes[2], otherproc);
103
104         bulkData.modification_end();
105
106         const size_t expectedTotalNumNodes = 4;
107         verify_global_node_count(expectedTotalNumNodes, bulkData);
108     }
109 }
```

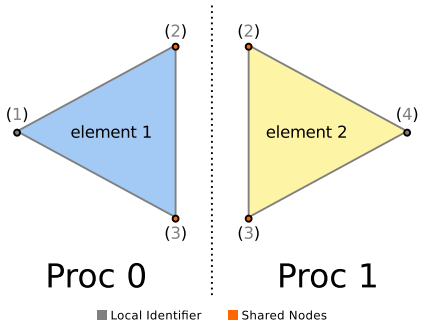

Figure 1.7: Creation of shared nodes for code listing 1.13

STK Mesh also supports the creation of independent shared nodes (nodes without connectivity) for use in p-refinement. In this case, additional nodes are created for higher order elements and these are maintained without explicit connectivity information in STK Mesh. Some of these nodes need to be shared across processor boundaries. This capability is to support the exploration of p-refinement. Currently, this capability cannot predict which nodes are attached to which elements when `change_entity_owner()` is called and therefore rebalance operations will likely not work as anticipated. This additional feature of `add_node_sharing()` is only enabled when the nodes are initially created. The code listing 1.14 demonstrates the use of `add_node_sharing()` to create independent shared nodes.

Listing 1.14: Example showing creation of independent shared nodes ../../../code/stk/stk_doc_tests/stk_mesh/createSharedNodes.cpp

```
113 TEST(stkMeshHowTo, createIndependentSharedNodes)
114 {
115     const unsigned spatialDimension = 2;
116     stk::mesh::MetaData metaData(spatialDimension, stk::mesh::entity_rank_names());
117     metaData.commit();
118
119     stk::mesh::BulkData bulkData(metaData, MPI_COMM_WORLD);
120     if (bulkData.parallel_size() == 2)
121     {
122         bulkData.modification_begin();
123
124         const unsigned nodesPerProc = 3;
125         std::vector<stk::mesh::EntityIdVector> nodeIds = { {1, 3, 2}, {4, 2, 3} };
126         const int myproc = bulkData.parallel_rank();
127         stk::mesh::EntityVector nodes(nodesPerProc);
128         nodes[0] = bulkData.declare_entity(stk::topology::NODE_RANK, nodeIds[myproc][0]);
129         nodes[1] = bulkData.declare_entity(stk::topology::NODE_RANK, nodeIds[myproc][1]);
130         nodes[2] = bulkData.declare_entity(stk::topology::NODE_RANK, nodeIds[myproc][2]);
```

```
131
132         int otherproc = testUtils::get_other_proc(myproc);
133         bulkData.add_node_sharing(nodes[1], otherproc);
134         bulkData.add_node_sharing(nodes[2], otherproc);
135
136         bulkData.modification_end();
137
138         const size_t expectedTotalNumNodes = 4;
139         verify_global_node_count(expectedTotalNumNodes, bulkData);
140     }
141 }
```

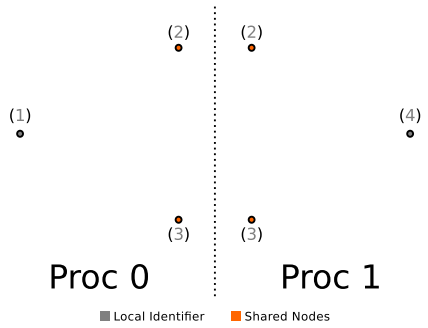

Figure 1.8: creation of independent shared nodes for code listing 1.14

This special marking to allow unconnected nodes to be shared will be removed if relations are attached to the node. The example 1.15 is a demonstration of this feature.

Listing 1.15: Example showing independent shared nodes becoming dependent../../../code/stk/stk_doc_tests/stk_mesh/createSharedNodes.cpp

```
145 TEST(stkMeshHowTo, createIndependentSharedNodesThenAddDependence)
146 {
147     const unsigned spatialDimension = 2;
148     stk::mesh::MetaData metaData(spatialDimension, stk::mesh::entity_rank_names());
149     stk::mesh::Part &triPart = metaData.declare_part_with_topology("triPart",
                stk::topology::TRIANGLE_3_2D);
150     metaData.commit();
151
152     stk::mesh::BulkData bulkData(metaData, MPI_COMM_WORLD);
153     if(bulkData.parallel_size() == 2)
154     {
155         bulkData.modification_begin();
156
157         const unsigned nodesPerProc = 3;
158         std::vector<stk::mesh::EntityIdVector> nodeIds = { {1, 3, 2}, {4, 2, 3}};
159         const int myproc = bulkData.parallel_rank();
160
161         stk::mesh::EntityVector nodes(nodesPerProc);
162         nodes[0] = bulkData.declare_entity(stk::topology::NODE_RANK, nodeIds[myproc][0]);
163         nodes[1] = bulkData.declare_entity(stk::topology::NODE_RANK, nodeIds[myproc][1]);
164         nodes[2] = bulkData.declare_entity(stk::topology::NODE_RANK, nodeIds[myproc][2]);
165
166         int otherproc = testUtils::get_other_proc(myproc);
167         bulkData.add_node_sharing(nodes[1], otherproc);
168         bulkData.add_node_sharing(nodes[2], otherproc);
169
170         const size_t expectedNumNodesPriorToModEnd = 6;
171         verify_global_node_count(expectedNumNodesPriorToModEnd, bulkData);
172
173         bulkData.modification_end();
```

```
174
175         const size_t expectedNumNodesAfterModEnd = 4; // nodes 2 and 3 are shared
176         verify_global_node_count(expectedNumNodesAfterModEnd, bulkData);
177
178         const unsigned elemsPerProc = 1;
179         stk::mesh::EntityId elemIds[][elemsPerProc] = { {1}, {2}};
180
181         bulkData.modification_begin();
182         stk::mesh::Entity elem = bulkData.declare_entity(stk::topology::ELEMENT_RANK,
                elemIds[myproc][0], triPart);
183         bulkData.declare_relation(elem, nodes[0], 0);
184         bulkData.declare_relation(elem, nodes[1], 1);
185         bulkData.declare_relation(elem, nodes[2], 2);
186         EXPECT_NO_THROW(bulkData.modification_end());
187
188         bulkData.modification_begin();
189         bulkData.destroy_entity(elem);
190         bulkData.modification_end();
191
192         if(myproc == 0)
193             verify_nodes_2_and_3_are_no_longer_shared(bulkData, nodes);
194
195         else  // myproc == 1
196             verify_nodes_2_and_3_are_removed(bulkData, nodes);
197     }
198 }
```

1.6.2.4 Change Entity Part Membership

`BulkData::change_entity_parts()` changes which *parts* an entity belongs to.

Changes in part membership can result in changes to "induced" part membership. (See Section 1.5.2.) Changes in part membership typically cause entities to move to different buckets.

1.6.2.5 Change Connectivity

`BulkData::declare_relation()` adds connectivity between two entities. `destroy_relation()` removes connectivity between two entities. Relations must be destroyed from the point of view of the higher-ranked entity toward the lower-ranked entity, although the relation in the other direction will also be removed automatically.

Changes in connectivity can result in changes to induced part membership. (See Section 1.5.2). Changes in connectivity can also result in changes in sharing and automatic ghosting during `modification_end()`. By causing changes in part membership(s), changes in connectivity can also result in changes to bucket structure.

1.6.2.6 Change Entity Ownership

In a parallel mesh, it can be necessary to change what processor rank owns an entity. The typical case is when there is a change to parallel decomposition.

The `change_entity_owner` method is used for this and is called with a vector of pairs that specify entities and destination processors. It must be called on all processes even if the input vector is empty on some processors.

Changes in ownership can cause changes in ghosting and sharing, which are changes to part membership. By causing changes in part membership(s), changes in ownership can also result in changes to bucket structure.

1.6.2.7 Change Ghosting

Aura ghosting is maintained automatically by STK Mesh, but can be optionally disabled. STK allows for application-specificed *custom ghosting*, through the functions `change_ghosting()`, `create_ghosting()`, `destroy_ghosting()`, and `destroy_all_ghosting()`. Each of these functions must be called parallel-synchronously.

The method `change_ghosting()` is used to add entities to be ghosted, or remove entities from a current ghosting. The input to the method includes a vector of pairs of entities and destination processors on which the entities are to be ghosted. To be added to a ghosting in this way, an entity must be locally-owned on the current processor, and must not already be shared by the destination processor. It is permissible for an entity to be in multiple different custom ghostings at the same time.

Any modification, directly applied or automatically called, to an entity in a ghosting will automatically cause that ghosting to be invalidated. For the aura ghosting, entities will be automatically regenerated during the next `modification_end()` call. For custom ghosting, it is not as well-defined what should happen to modified entities. It is possible for an entity in a ghosting to be invalidated without all of that ghosting being invalidated. `stk::mesh::BulkData::is_valid(entity)` can be used to determine whether a ghost entity has been invalidated.

1.6.3 Mesh Modification Examples

Listing 1.16 shows how an element on processor 0 in the mesh depicted in Figure 1.9 is ghosted to processor 1. Note that Element 1 is connected to Node 1. This test shows how a user can use the identifier of the element, i.e. 1, to get an entity, and ghost it to another processor. This test also shows that Node 1 is automatically ghosted to processor 1 because it is a downward-relation of Element 1. In general, when an entity is ghosted, its downward-connected entities come along with it, but upward-connected entities don't.

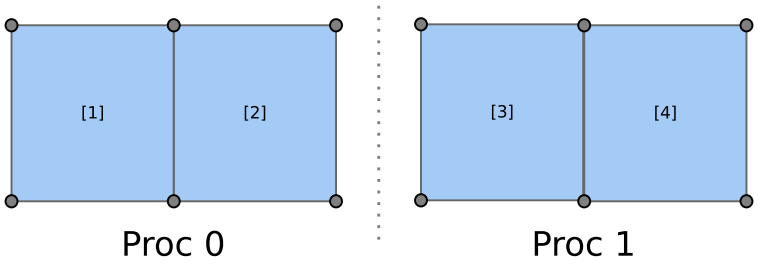

Figure 1.9: Mesh Used in Listings 1.16-1.17

Listing 1.16: Example showing an element being ghosted.
../../../code/stk/stk_doc_tests/stk_mesh/customGhosting.cpp

```
95  TEST(StkMeshHowTo, customGhostElem)
96  {
97      MPI_Comm communicator = MPI_COMM_WORLD;
98      if (stk::parallel_machine_size(communicator) == 2)
99      {
100         stk::mesh::MetaData metaData;
101         stk::mesh::BulkData bulkData(metaData, communicator);
102         stk::unit_test_util::fill_mesh_using_stk_io("generated:1x1x4", bulkData);
103
104         stk::mesh::EntityId id = 1;
105         stk::mesh::Entity elem1 = bulkData.get_entity(stk::topology::ELEM_RANK, id);
106         stk::mesh::Entity node1 = bulkData.get_entity(stk::topology::NODE_RANK, id);
107         verify_that_elem1_and_node1_are_only_valid_on_p0(bulkData, elem1, node1);
108
109         bulkData.modification_begin();
110         stk::mesh::Ghosting& ghosting = bulkData.create_ghosting("custom ghost for elem 1");
111         std::vector<std::pair<stk::mesh::Entity, int> > elemProcPairs;
112         if (bulkData.parallel_rank() == 0)
113           elemProcPairs.push_back(std::make_pair(elem1,
              get_other_proc(bulkData.parallel_rank())));
114         bulkData.change_ghosting(ghosting, elemProcPairs);
115         bulkData.modification_end();
116
117         verify_that_elem1_and_downward_connected_entities_are_ghosted_from_p0_to_p1(bulkData,
              id);
118     }
119 }
120
121 TEST(StkMeshHowTo, addElementToGhostingUsingSpecializedModificationForPerformance)
122 {
123     MPI_Comm communicator = MPI_COMM_WORLD;
124     if(stk::parallel_machine_size(communicator) == 2)
125     {
126         stk::mesh::MetaData meta;
127         stk::mesh::BulkData bulk(meta, communicator);
128         stk::unit_test_util::fill_mesh_using_stk_io("generated:1x1x4", bulk);
129
130         stk::mesh::EntityId elementId = 1;
131         stk::mesh::Entity elem1 = bulk.get_entity(stk::topology::ELEM_RANK, elementId);
132         verify_elem1_is_valid_only_on_p0(bulk, elem1);
133
134         bulk.modification_begin();
135         stk::mesh::Ghosting& ghosting = bulk.create_ghosting("my custom ghosting");
136         bulk.modification_end();
137
138         stk::mesh::EntityProcVec entityProcPairs;
139         if(bulk.parallel_rank() == 0)
140             entityProcPairs.push_back(stk::mesh::EntityProc(elem1,
                 get_other_proc(bulk.parallel_rank())));
141
```

```
142         bulk.batch_add_to_ghosting(ghosting, entityProcPairs);
143
144         verify_elem1_is_valid_on_both_procs(bulk, elementId);
145     }
146 }
```

Listing 1.17 shows how an entity can be moved, or stated alternatively, how to change an owner of an entity. Note that the `change_entity_owner()` method must be called by all processors, and must not be enclosed within calls to `modification_begin()` and `modification_end()` since it is a self-contained modification cycle.

Listing 1.17: Example of changing processor ownership of an element ../../../code/stk/stk_doc_tests/stk_mesh/changeEntityOwner.cpp

```
66 TEST(StkMeshHowTo, changeEntityOwner)
67 {
68     MPI_Comm communicator = MPI_COMM_WORLD;
69     if (stk::parallel_machine_size(communicator) == 2)
70     {
71         stk::mesh::MetaData metaData;
72         stk::mesh::BulkData bulkData(metaData, communicator);
73         stk::unit_test_util::fill_mesh_using_stk_io("generated:1x1x4", bulkData);
74
75         stk::mesh::EntityId elem2Id = 2;
76         stk::mesh::Entity elem2 = bulkData.get_entity(stk::topology::ELEM_RANK, elem2Id);
77         verify_elem_is_owned_on_p0_and_valid_as_aura_on_p1(bulkData, elem2);
78
79         std::vector<std::pair<stk::mesh::Entity, int> > elemProcPairs;
80         if (bulkData.parallel_rank() == 0)
81           elemProcPairs.push_back(std::make_pair(elem2,
              testUtils::get_other_proc(bulkData.parallel_rank())));
82
83         bulkData.change_entity_owner(elemProcPairs);
84
85         verify_elem_is_now_owned_on_p1(bulkData, elem2Id);
86     }
87 }
```

1.6.3.1 Resolving Sharing Of Exodus Sidesets - Special Case

Figure 1.10 shows a case of an interior Exodus sideset where two sides exist initially across a processor boundary. Nodes (1, 5, 8, 4) represent the face on the left (red) element on processor 0, and the nodes (1, 4, 8, 5) represent the face on the right (green) element on processor 1. The algorithm for determining if these two faces are the same shared face will consider the following two conditions:

1. The nodes on both face entities are the same or a valid permutation of each other
2. The identifiers of both face entities are the same

A boolean flag exists on BulkData, that if set to true, will require that two entities are the same if both conditions, (1) and (2), must be true for the entity to be marked as shared.

When reading an Exodus file and populating a STK Mesh, the current setting is that both conditions must be true for the mesh entities to be marked as the same. However, after the mesh has been read in, only condition (1) is used to resolve sharing of entities across parallel boundaries.

If the user desires one behavior over another, the `set_use_entity_ids_for_resolving_sharing()` function can be used before calling `modification_end()` during a mesh modification cycle. This behavior is undergoing changes so that the face entities created are consistently connected to elements. As such, the option discussed here is marked to be deprecated.

Code listing 1.18 shows two tests. The first test shows the option that can be used for resolving sharing. The second test case reads the mesh in Figure 1.10 and tests that there are two faces.

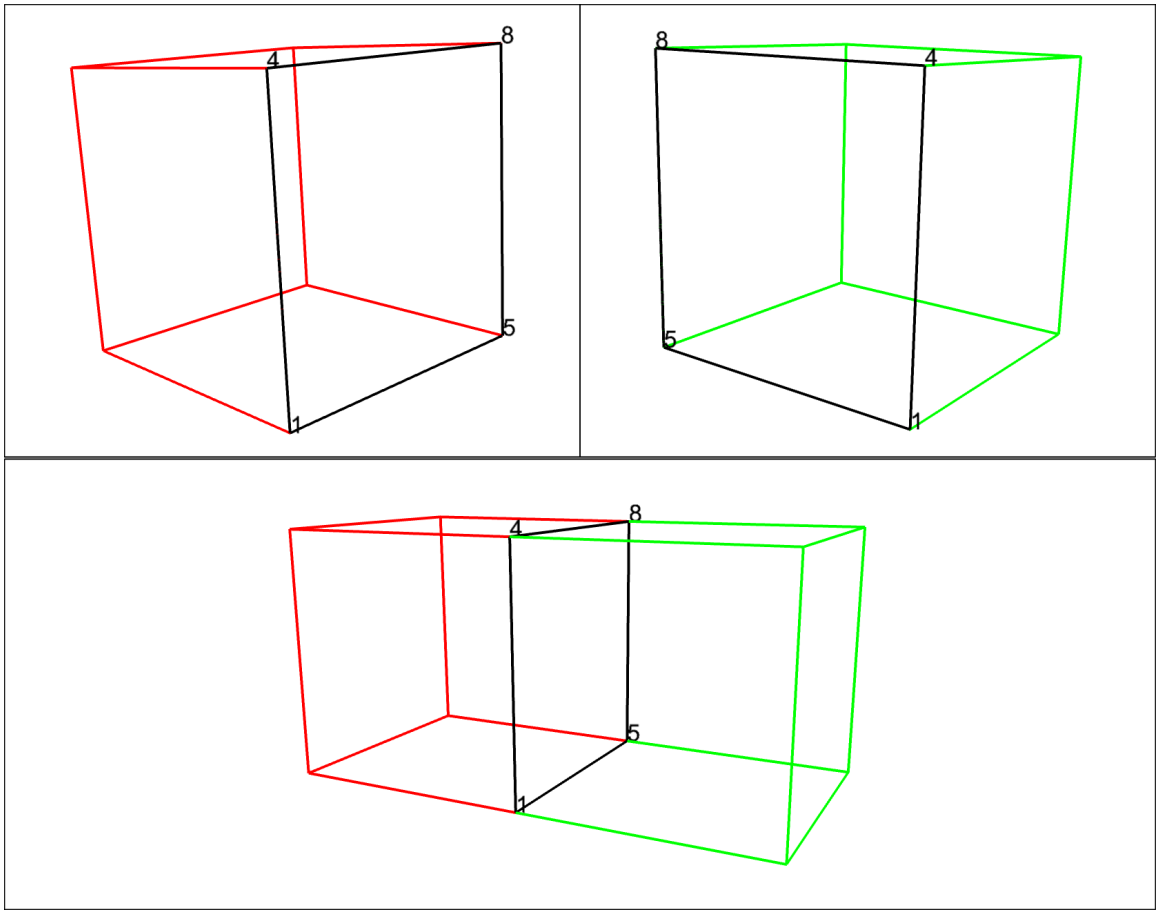

Figure 1.10: Mesh Used in Listing 1.18

Listing 1.18: Example of internal sideset which results in two faces
../../../code/stk/stk_integration_tests/stk_mesh_doc/IntegrationTestBulkData.cpp

```
80 TEST(BulkData_test, use_entity_ids_for_resolving_sharing)
81 {
82     MPI_Comm communicator = MPI_COMM_WORLD;
83
84     const int spatialDim = 3;
85     stk::mesh::MetaData stkMeshMetaData(spatialDim);
```

```
86      stk::unit_test_util::BulkDataTester stkMeshBulkData(stkMeshMetaData, communicator);
87
88      if(stkMeshBulkData.parallel_size() == 2)
89      {
90          std::string exodusFileName = unitTestUtils::getOption("-i", "mesh.exo");
91
92          {
93              stk::io::StkMeshIoBroker exodusFileReader(communicator);
94              exodusFileReader.set_bulk_data(stkMeshBulkData);
95              exodusFileReader.add_mesh_database(exodusFileName, stk::io::READ_MESH);
96              exodusFileReader.create_input_mesh();
97              exodusFileReader.populate_bulk_data();
98          }
99      }
100
101     stkMeshBulkData.set_use_entity_ids_for_resolving_sharing(false);
102     EXPECT_FALSE(stkMeshBulkData.use_entity_ids_for_resolving_sharing());
103
104     stkMeshBulkData.set_use_entity_ids_for_resolving_sharing(true);
105     EXPECT_TRUE(stkMeshBulkData.use_entity_ids_for_resolving_sharing());
106 }
107
108 TEST(BulkData_test, testTwoDimProblemForSharingOfDifferentEdgesWithSameNodesFourProc)
109 {
110     MPI_Comm communicator = MPI_COMM_WORLD;
111     const int spatialDim = 2;
112     stk::mesh::MetaData stkMeshMetaData(spatialDim);
113     stk::unit_test_util::BulkDataTester stkMeshBulkData(stkMeshMetaData, communicator);
114
115     if ( stkMeshBulkData.parallel_size() == 4 )
116     {
117         std::string exodusFileName = unitTestUtils::getOption("-i", "mesh.exo");
118
119         {
120             stk::io::StkMeshIoBroker exodusFileReader(communicator);
121             exodusFileReader.set_bulk_data(stkMeshBulkData);
122             exodusFileReader.add_mesh_database(exodusFileName, stk::io::READ_MESH);
123             exodusFileReader.create_input_mesh();
124             // With in populate_bulk_data, the option
                 set_use_entity_ids_for_resolving_sharing is set to true
125             exodusFileReader.populate_bulk_data();
126         }
127
128         std::vector<size_t> globalCounts;
129         stk::mesh::comm_mesh_counts(stkMeshBulkData, globalCounts);
130         EXPECT_EQ(15u, globalCounts[stk::topology::EDGE_RANK]);
131     }
132 }
```

1.6.4 Unsafe operations

There are a number of operations that are inherently unsafe to perform when the mesh is in the middle of a modification cycle. Exceptions will be thrown if the user tries to perform these operations during modification in a debug build, but not in a release build since the error checking is too expensive.

The *mesh_index* of an entity (which is a pairing of the entity's bucket and the entity's offset into that bucket) can be automatically changed by STK Mesh during a modification cycle. Thus, a mesh_index cannot be assumed to be valid during a modifcation cycle or be the same before and

after it. A change in the membership of one or more buckets implies a change in the mesh index of one or more entities, and vice versa.

Although field data can be accessed during a modification cycle, parallel field operations (e.g., parallel sum) must be avoided during a modification cycle because the status of parallel sharing is not guaranteed to be globally consistent until after `BulkData::modification_end()`.

Mesh modification should generally not be done while looping over buckets. The problem is that mesh modification can cause entities to move from one bucket to another, which can invalidate the iteration over a particular bucket. Any loop that makes the assumption of Bucket stability, either the existence/order of a Bucket or the order of entities within the bucket, is not safe if the loop does mesh modification. Some errors that can result will be checked in debug, but never in release. If you must iterate the mesh and do mesh modification during the iteration, use an entity loop, not a bucket loop.

1.6.5 Automatic modification operations in modification_end()

When the client code is finished with all direct calls to any of the modifications in Section 1.6.2, it must call `modification_end()` to close the modification cycle.

`BulkData::modification_end()` automatically performs several types of modifications to the mesh to bring it into a parallel consistent state. These include

- Synchronizing entity membership in parts for shared entities.
- Refreshing the ghost layer around shared entities (referred to as the aura).
- Updating ghost entities in the aura that have changed part membership.
- Sorting buckets' entities for a well-defined ordering.
- Resolve side creation on the subdomain boundaries.

It is important to note that `modification_end()` used to automatically determine the sharing of nodes that had been created with the same global identifier on multiple MPI processors. It no longer does this, and client code is now required to inform STK Mesh of node sharing information. See section 1.6.2.3 for more details.

Since the sharing of entities is only changed automatically by STK Mesh internally, that functionality is not available through the STK Mesh API.

1.6.6 How to use generate_new_entities()

This example (Listing 1.19) shows how to use `BulkData::generate_new_entities()` to create new entities. After the entities are created, the ELEMENT_RANK entities are each assigned a topology and their nodal relations are set before `BulkData::modification_end()` is called. FACE_RANK and EDGE_RANK entities have the same requirement, but none

are included in this example. The example also illustrates that it is incorrect to call `BulkData::modification_end()` if the requirement is not met.

Listing 1.19: Example of how to generate multiple new entities and subsequently set topologies and nodal relations ../../../code/stk/stk_mesh/testsForDocumentation/generateNewEntities.cpp

```
68 TEST(stkMeshHowTo, generateNewEntities)
69 {
70   const unsigned spatialDimension = 3;
71
72   stk::mesh::MetaData metaData(spatialDimension, stk::mesh::entity_rank_names());
73   stk::mesh::Part &tetPart = metaData.declare_part_with_topology("tetElementPart",
             stk::topology::TET_4);
74   stk::mesh::Part &hexPart = metaData.declare_part_with_topology("hexElementPart",
             stk::topology::HEX_8);
75   metaData.commit();
76
77   // Parts vectors handy for setting topology later.
78   std::vector<stk::mesh::Part *> add_tetPart(1);
79   add_tetPart[0] = &tetPart;
80   std::vector<stk::mesh::Part *> add_hexPart(1);
81   add_hexPart[0] = &hexPart;
82
83   stk::mesh::BulkData mesh(metaData, MPI_COMM_WORLD);
84   mesh.modification_begin();
85
86   std::vector<size_t> requests(metaData.entity_rank_count(), 0);
87   const size_t num_nodes_requested = 12;
88   const size_t num_elems_requested =  2;
89   requests[stk::topology::NODE_RANK] = num_nodes_requested;
90   requests[stk::topology::ELEMENT_RANK] = num_elems_requested;
91   std::vector<stk::mesh::Entity> requested_entities;
92
93   mesh.generate_new_entities(requests, requested_entities);
94
95   // Set topologies of new entities with rank > stk::topology::NODE_RANK.
96   stk::mesh::Entity elem1 = requested_entities[num_nodes_requested];
97   mesh.change_entity_parts(elem1, add_tetPart);
98   stk::mesh::Entity elem2 = requested_entities[num_nodes_requested + 1];
99   mesh.change_entity_parts(elem2, add_hexPart);
100
101  // Set downward relations of entities with rank > stk::topology::NODE_RANK
102  unsigned node_i = 0;
103  for(unsigned node_ord = 0 ; node_ord < 4; ++node_ord, ++node_i)
104  {
105    mesh.declare_relation( elem1 , requested_entities[node_i] , node_ord);
106  }
107  for(unsigned node_ord = 0 ; node_ord < 8; ++node_ord, ++node_i)
108  {
109    mesh.declare_relation( elem2 , requested_entities[node_i] , node_ord);
110  }
111  mesh.modification_end();
112
113  check_connectivities_for_stkMeshHowTo_generateNewEntities(mesh, elem1, elem2,
             requested_entities);
114
115  // Not setting topologies of new entities with rank > stk::topology::NODE_RANK causes throw
116  mesh.modification_begin();
117  std::vector<stk::mesh::Entity> more_requested_entities;
118  mesh.generate_new_entities(requests, more_requested_entities);
119 #ifdef NDEBUG
120  mesh.modification_end();
121 #else
122  EXPECT_THROW(mesh.modification_end(), std::logic_error);
123 #endif
124 }
```

1.6.7 How to create faces

STK Mesh provides functions for creating all edges or faces for an existing mesh. This example demonstrates first creating a mesh of hex elements with nodes, (generated by STK IO), then uses the `create_faces()` function to create all faces in the mesh.

Listing 1.20: Example of how to create all element faces
../../../code/stk/stk_mesh/testsForDocumentation/createFacesHex.cpp

```
49  TEST(StkMeshHowTo, CreateFacesHex)
50  {
51    // ============================================================
52    // INITIALIZATION
53    MPI_Comm communicator = MPI_COMM_WORLD;
54    if (stk::parallel_machine_size(communicator) != 1) { return; }
55    stk::io::StkMeshIoBroker stkIo(communicator);
56
57    const std::string generatedFileName = "generated:8x8x8";
58    stkIo.add_mesh_database(generatedFileName, stk::io::READ_MESH);
59    stkIo.create_input_mesh();
60    stkIo.populate_bulk_data();
61
62    // ============================================================
63    //+ EXAMPLE
64    //+ Create the faces..
65    stk::mesh::create_faces(stkIo.bulk_data());
66
67    // ============================================================
68    // VERIFICATION
69    stk::mesh::Selector allEntities = stkIo.meta_data().universal_part();
70    std::vector<unsigned> entityCounts;
71    stk::mesh::count_entities(allEntities, stkIo.bulk_data(), entityCounts);
72    EXPECT_EQ( 512u, entityCounts[stk::topology::ELEMENT_RANK]);
73    EXPECT_EQ(1728u, entityCounts[stk::topology::FACE_RANK]);
74
75    // Edges are not generated, only faces.
76    EXPECT_EQ(0u,   entityCounts[stk::topology::EDGE_RANK]);
77  }
```

1.6.8 How to create both edges and faces

This example demonstrates create all edges as well as faces for a hex-element mesh. Note that these functions only create relations to elements and nodes, so the faces will not have relations to the edges when both `create_edges()` and `create_faces()` are called.

Listing 1.21: Example of how to create all element edges and faces
../../../code/stk/stk_mesh/testsForDocumentation/createFacesEdgesHex.cpp

```
61  TEST(StkMeshHowTo, CreateFacesEdgesHex)
62  {
63    // ============================================================
64    // INITIALIZATION
65    MPI_Comm communicator = MPI_COMM_WORLD;
66    if (stk::parallel_machine_size(communicator) != 1) { return; }
67    stk::io::StkMeshIoBroker stkIo(communicator);
68
69    const std::string generatedFileName = "generated:8x8x8";
70    stkIo.add_mesh_database(generatedFileName, stk::io::READ_MESH);
```

```
71      stkIo.create_input_mesh();
72      stkIo.populate_bulk_data();
73
74      // ============================================================
75      //+ EXAMPLE
76      //+ Create the faces..
77      stk::mesh::create_faces(stkIo.bulk_data());
78
79      //+ Create the edges..
80      stk::mesh::create_edges(stkIo.bulk_data());
81
82      // ============================================================
83      // VERIFICATION
84      stk::mesh::Selector allEntities = stkIo.meta_data().universal_part();
85      std::vector<unsigned> entityCounts;
86      stk::mesh::count_entities(allEntities, stkIo.bulk_data(), entityCounts);
87      EXPECT_EQ( 512u, entityCounts[stk::topology::ELEMENT_RANK]);
88      EXPECT_EQ(1728u, entityCounts[stk::topology::FACE_RANK]);
89      EXPECT_EQ(1944u, entityCounts[stk::topology::EDGE_RANK]);
90      // MAKE SURE FACES ARE HOOKED TO EDGES
91      // this should happen if create_faces is called before create_edges
92      stk::mesh::BucketVector const & face_buckets =
                stkIo.bulk_data().buckets(stk::topology::FACE_RANK);
93      for (size_t bucket_count=0, bucket_end=face_buckets.size(); bucket_count < bucket_end;
                ++bucket_count) {
94        stk::mesh::Bucket & bucket = *face_buckets[bucket_count];
95        const unsigned num_expected_edges = bucket.topology().num_edges();
96        EXPECT_EQ(4u, num_expected_edges);
97        for (size_t face_count=0, face_end=bucket.size(); face_count < face_end; ++face_count) {
98          stk::mesh::Entity face = bucket[face_count];
99          EXPECT_EQ(num_expected_edges, stkIo.bulk_data().num_edges(face));
100       }
101     }
102   }
```

1.6.9 How to create faces on only selected elements

This example demonstrates creating faces for a subset of the mesh elements defined by a Selector. Note that the "generated-mesh" syntax specifies that the initial mesh contains not only hex elements but also shell elements on all 6 sides.

Listing 1.22: Example of how to create faces on only selected elements
../../../code/stk/stk_mesh/testsForDocumentation/createSelectedFaces.cpp

```
52  TEST(StkMeshHowTo, CreateSelectedFacesHex)
53  {
54    // ============================================================
55    // INITIALIZATION
56    MPI_Comm communicator = MPI_COMM_WORLD;
57    if (stk::parallel_machine_size(communicator) != 1) { return; }
58    stk::io::StkMeshIoBroker stkIo(communicator);
59
60    // Generate a mesh containing 1 hex part and 6 shell parts
61    const std::string generatedFileName = "generated:8x8x8|shell:xyzXYZ";
62    stkIo.add_mesh_database(generatedFileName, stk::io::READ_MESH);
63    stkIo.create_input_mesh();
64    stkIo.populate_bulk_data();
65    const stk::mesh::PartVector &all_parts = stkIo.meta_data().get_mesh_parts();
66
67    // ============================================================
68    //+ EXAMPLE
```

```
69      //+ Create a selector containing just the shell parts.
70      stk::mesh::Selector shell_subset;
71      for (size_t i=0; i < all_parts.size(); i++) {
72        const stk::mesh::Part *part = all_parts[i];
73        stk::topology topo = part->topology();
74        if (topo == stk::topology::SHELL_QUAD_4) {
75          shell_subset |= *part;
76        }
77      }
78
79      //+ Create the faces on just the selected shell parts.
80      stk::mesh::create_faces(stkIo.bulk_data(), shell_subset);
81
82      // ============================================
83      // VERIFICATION
84      stk::mesh::Selector allEntities = stkIo.meta_data().universal_part();
85      std::vector<unsigned> entityCounts;
86      stk::mesh::count_entities(allEntities, stkIo.bulk_data(), entityCounts);
87      EXPECT_EQ( 896u, entityCounts[stk::topology::ELEMENT_RANK]);
88      EXPECT_EQ( 768u, entityCounts[stk::topology::FACE_RANK]);
89
90      // Edges are not generated, only faces.
91      EXPECT_EQ(0u,   entityCounts[stk::topology::EDGE_RANK]);
92    }
93
```

1.6.10 Creating faces with layered shells

This example shows how many faces will be created when there are layered shells present.

Listing 1.23: Example showing that faces are created correctly when layered shells are present ../../../code/stk/stk_mesh/testsForDocumentation/CreateFacesLayeredShellsHex.cpp

```
49    TEST(StkMeshHowTo, CreateFacesLayeredShellsHex)
50    {
51      // ============================================================
52      // INITIALIZATION
53      MPI_Comm communicator = MPI_COMM_WORLD;
54      if (stk::parallel_machine_size(communicator) != 1) { return; }
55      stk::io::StkMeshIoBroker stkIo(communicator);
56
57      // Generate a mesh containing 1 hex part and 12 shell parts
58      // Shells are layered 2 deep.
59      const std::string generatedFileName = "generated:8x8x8|shell:xxyyzzXYZXYZ";
60      stkIo.add_mesh_database(generatedFileName, stk::io::READ_MESH);
61      stkIo.create_input_mesh();
62      stkIo.populate_bulk_data();
63
64      // ============================================================
65      //+ EXAMPLE
66      //+ Create the faces
67      stk::mesh::create_faces(stkIo.bulk_data());
68
69      // ============================================
70      // VERIFICATION
71      stk::mesh::Selector allEntities = stkIo.meta_data().universal_part();
72      std::vector<unsigned> entityCounts;
73      stk::mesh::count_entities(allEntities, stkIo.bulk_data(), entityCounts);
74      EXPECT_EQ(1280u, entityCounts[stk::topology::ELEMENT_RANK]);
75      //+ The shell faces are the same as the boundary hex faces
76      EXPECT_EQ(2112u, entityCounts[stk::topology::FACE_RANK]);
77
```

```
78      // Edges are not generated, only faces.
79      EXPECT_EQ(0u,   entityCounts[stk::topology::EDGE_RANK]);
80    }
81
```

1.6.11 Creating faces between hexes, on shells, and on shells between hexes

This example shows how many faces are created on interior faces between hexes and shells.

Listing 1.24: Example of how many faces get constructed by CreateFaces between two hexes. ../../../code/stk/stk_mesh/testsForDocumentation/CreateFacesHexesShells.cpp

```
52 TEST(StkMeshHowTo, CreateFacesTwoHexes)
53 {
54     if (stk::parallel_machine_size(MPI_COMM_WORLD) == 1) {
55         //  -----------
56         //  |    |    |
57         //  |HEX1|HEX2|
58         //  |    |    |
59         //  -----------
60         stk::io::StkMeshIoBroker stkMeshIoBroker(MPI_COMM_WORLD);
61         stkMeshIoBroker.add_mesh_database("AA.e", stk::io::READ_MESH);
62         stkMeshIoBroker.create_input_mesh();
63         stkMeshIoBroker.populate_bulk_data();
64         stk::mesh::BulkData &mesh = stkMeshIoBroker.bulk_data();
65
66         stk::mesh::create_faces(mesh);
67
68         //  ------  F  ------
69         //  |    |  A  |    |
70         //  |HEX1|<-C->|HEX2|   Also external faces!
71         //  |    |  E  |    |
72         //  ------  !  ------
73
74         unsigned first_bucket = 0;
75         unsigned first_element_in_bucket = 0;
76         stk::mesh::Entity first_element =
               (*mesh.buckets(stk::topology::ELEMENT_RANK)[first_bucket])[first_element_in_bucket];
77         stk::mesh::Entity internal_face = mesh.begin_faces(first_element)[5];
78
79         unsigned num_elements_connected_to_single_face = 2;
80         EXPECT_EQ(num_elements_connected_to_single_face, mesh.num_elements(internal_face));
81
82         unsigned num_expected_external_faces = 10u;
83         unsigned num_expected_internal_faces = 1u;
84         unsigned num_expected_faces = num_expected_external_faces +
               num_expected_internal_faces;
85         stk::mesh::Selector all_entities = mesh.mesh_meta_data().universal_part();
86         std::vector<unsigned> entity_counts;
87         stk::mesh::count_entities(all_entities, mesh, entity_counts);
88         EXPECT_EQ(num_expected_faces, entity_counts[stk::topology::FACE_RANK]);
89     }
90 }
```

Listing 1.25: Example of how many faces get constructed by CreateFaces on a shell. ../../../code/stk/stk_mesh/testsForDocumentation/CreateFacesHexesShells.cpp

```
94 TEST(StkMeshHowTo, CreateFacesSingleShell)
95 {
96     if (stk::parallel_machine_size(MPI_COMM_WORLD) == 1) {
```

```
97          //  S
98          //  H
99          //  E
100         //  L
101         //  L
102         stk::io::StkMeshIoBroker stkMeshIoBroker(MPI_COMM_WORLD);
103         stkMeshIoBroker.add_mesh_database("e.e", stk::io::READ_MESH);
104         stkMeshIoBroker.create_input_mesh();
105         stkMeshIoBroker.populate_bulk_data();
106         stk::mesh::BulkData &mesh = stkMeshIoBroker.bulk_data();
107
108         stk::mesh::create_faces(mesh);
109
110         //  F  S  F
111         //  A  H  A
112         //  C->E<-C
113         //  E  L  E
114         //  1  L  2
115
116         unsigned first_bucket = 0;
117         unsigned first_element_in_bucket = 0;
118         stk::mesh::Entity first_element =
                (*mesh.buckets(stk::topology::ELEMENT_RANK)[first_bucket])[first_element_in_bucket];
119         stk::mesh::Entity face_one = mesh.begin_faces(first_element)[0];
120         unsigned num_elements_connected_to_face_one = 1;
121         EXPECT_EQ(num_elements_connected_to_face_one, mesh.num_elements(face_one));
122
123         stk::mesh::Entity face_two = mesh.begin_faces(first_element)[1];
124         unsigned num_elements_connected_to_face_two = 1;
125         EXPECT_EQ(num_elements_connected_to_face_two, mesh.num_elements(face_two));
126
127         EXPECT_NE(face_one, face_two);
128
129         unsigned num_expected_faces = 2u;
130         stk::mesh::Selector all_entities = mesh.mesh_meta_data().universal_part();
131         std::vector<unsigned> entity_counts;
132         stk::mesh::count_entities(all_entities, mesh, entity_counts);
133         EXPECT_EQ(num_expected_faces, entity_counts[stk::topology::FACE_RANK]);
134     }
135 }
```

Listing 1.26: Example of how many faces get constructed by CreateFaces between hexes and an internal shell. ../../../code/stk/stk_mesh/testsForDocumentation/CreateFacesHexesShells.cpp

```
139 TEST(StkMeshHowTo, CreateFacesTwoHexesInternalShell)
140 {
141     if (stk::parallel_machine_size(MPI_COMM_WORLD) == 1) {
142         //  ------S------
143         //  |     |H|     |
144         //  |HEX1|E|HEX2|
145         //  |     |L|     |
146         //  ------L------
147         stk::io::StkMeshIoBroker stkMeshIoBroker(MPI_COMM_WORLD);
148         stkMeshIoBroker.add_mesh_database("AeA.e", stk::io::READ_MESH);
149         stkMeshIoBroker.create_input_mesh();
150         stkMeshIoBroker.populate_bulk_data();
151         stk::mesh::BulkData &mesh = stkMeshIoBroker.bulk_data();
152
153         stk::mesh::create_faces(mesh);
154
155         //  ------  F  S  F  ------
156         //  |     |  A  H  A  |     |
157         //  |HEX1|<-C->E<-C->|HEX2|   Also external faces!
158         //  |     |  E  L  E  |     |
159         //  ------  1  L  2  ------
160
```

```
161         unsigned first_bucket = 0;
162         unsigned first_element_in_bucket = 0;
163         stk::mesh::Entity first_element =
                (*mesh.buckets(stk::topology::ELEMENT_RANK)[first_bucket])[first_element_in_bucket];
164         stk::mesh::Entity internal_face_one = mesh.begin_faces(first_element)[5];
165         unsigned num_elements_connected_to_face_one = 2;
166         EXPECT_EQ(num_elements_connected_to_face_one, mesh.num_elements(internal_face_one));
167
168         unsigned second_element_in_bucket = 1;
169         stk::mesh::Entity second_element =
                (*mesh.buckets(stk::topology::ELEMENT_RANK)[first_bucket])[second_element_in_bucket];
170         stk::mesh::Entity internal_face_two = mesh.begin_faces(second_element)[4];
171         unsigned num_elements_connected_to_face_two = 2;
172         EXPECT_EQ(num_elements_connected_to_face_two, mesh.num_elements(internal_face_two));
173
174         EXPECT_NE(internal_face_one, internal_face_two);
175
176         unsigned num_expected_external_faces = 10u;
177         unsigned num_expected_internal_faces = 2u;
178         unsigned num_expected_faces = num_expected_external_faces +
                num_expected_internal_faces;
179         stk::mesh::Selector all_entities = mesh.mesh_meta_data().universal_part();
180         std::vector<unsigned> entity_counts;
181         stk::mesh::count_entities(all_entities, mesh, entity_counts);
182         EXPECT_EQ(num_expected_faces, entity_counts[stk::topology::FACE_RANK]);
183     }
184 }
```

1.6.12 How to skin a mesh

STK Mesh provides functions for skinning an existing mesh and creating appropriate boundary sides. This example demonstrates first creating a mesh of one hex element with nodes, (generated by STK IO), then uses the the `create_exposed_boundary_sides()` function to skin the mesh.

Listing 1.27: Example of how to create all the exposed boundary sides ../../../code/stk/stk_mesh/testsForDocumentation/howToSkinMesh.cpp

```
50 TEST(StkMeshHowTo, SkinExposedHex)
51 {
52     // ============================================================
53     // INITIALIZATION
54     MPI_Comm communicator = MPI_COMM_WORLD;
55     if (stk::parallel_machine_size(communicator) != 1) { return; }
56     stk::io::StkMeshIoBroker stkIo(communicator);
57
58     const std::string generatedFileName = "generated:1x1x1";
59     stkIo.add_mesh_database(generatedFileName, stk::io::READ_MESH);
60     stkIo.create_input_mesh();
61     stkIo.populate_bulk_data();
62
63     // ============================================================
64     //+ EXAMPLE
65     //+ Skin the mesh and create the exposed boundary sides..
66     stk::mesh::MetaData &metaData = stkIo.meta_data();
67     stk::mesh::BulkData &bulkData = stkIo.bulk_data();
68     stk::mesh::Selector allEntities = metaData.universal_part();
69     stk::mesh::Part &skinPart = metaData.declare_part("skin", metaData.side_rank());
70     stk::io::put_io_part_attribute(skinPart);
71
```

```
72      stk::mesh::create_exposed_block_boundary_sides(bulkData, allEntities, {&skinPart});
73
74      // ================================================
75      // VERIFICATION
76      EXPECT_TRUE(stk::mesh::check_exposed_block_boundary_sides(bulkData, allEntities,
                skinPart));
77      stk::mesh::Selector skin(skinPart & metaData.locally_owned_part());
78      unsigned numSkinnedSides = stk::mesh::count_selected_entities(skin,
                bulkData.buckets(metaData.side_rank()));
79      EXPECT_EQ(6u, numSkinnedSides) << "in part " << skinPart.name();
80  }
```

1.6.13 How to create internal block boundaries of a mesh

STK Mesh also provides functions for creating the interior block boundary sides of an existing mesh. This example demonstrates first creating a mesh of two hex element with nodes, (generated by STK IO), creation of an IOPart into which element 2 is moved, followed by `create_interior_block_boundary_sides()` function to skin the mesh interior.

Listing 1.28: Example of how to create all the interior block boundary sides ../../../code/stk/stk_mesh/testsForDocumentation/howToSkinMesh.cpp

```
84  TEST(StkMeshHowTo, SkinInteriorHex)
85  {
86      // ============================================================
87      // INITIALIZATION
88      MPI_Comm communicator = MPI_COMM_WORLD;
89      if (stk::parallel_machine_size(communicator) != 1) { return; }
90      stk::io::StkMeshIoBroker stkIo(communicator);
91
92      const std::string generatedFileName = "generated:1x1x2";
93      stkIo.add_mesh_database(generatedFileName, stk::io::READ_MESH);
94      stkIo.create_input_mesh();
95      stkIo.populate_bulk_data();
96
97      // ============================================================
98      //+ EXAMPLE
99      //+ Skin the mesh and create the exposed boundary sides..
100     stk::mesh::MetaData &metaData = stkIo.meta_data();
101     stk::mesh::BulkData &bulkData = stkIo.bulk_data();
102     stk::mesh::Selector allEntities = metaData.universal_part();
103     stk::mesh::Part &skinPart = metaData.declare_part("skin", metaData.side_rank());
104     stk::io::put_io_part_attribute(skinPart);
105
106     stk::mesh::Entity elem2 = bulkData.get_entity(stk::topology::ELEM_RANK, 2u);
107     stk::mesh::Part *block_1 = metaData.get_part("block_1");
108
109     bulkData.modification_begin();
110     stk::mesh::Part &block_2 = metaData.declare_part("block_2", stk::topology::ELEM_RANK);
111     stk::io::put_io_part_attribute(block_2);
112     bulkData.change_entity_parts(elem2, {&block_2}, {block_1});
113     bulkData.modification_end();
114
115     stk::mesh::create_interior_block_boundary_sides(bulkData, allEntities, {&skinPart});
116
117     // ================================================
118     // VERIFICATION
119     EXPECT_TRUE(stk::mesh::check_interior_block_boundary_sides(bulkData, allEntities,
                skinPart));
120     stk::mesh::Selector skin(skinPart & metaData.locally_owned_part());
```

```
121     unsigned numSkinnedSides = stk::mesh::count_selected_entities(skin,
                bulkData.buckets(metaData.side_rank()));
122     EXPECT_EQ(1u, numSkinnedSides) << "in part " << skinPart.name();
123 }
```

1.6.14 How to destroy elements in list

STK Mesh now provides a means by which an application may destroy all the elements in a list as well as the downward connected entities in order to ensure that there are no orphaned nodes/faces.

Listing 1.29: Example of how to destroy elements in a list ../../../code/stk/stk_mesh/testsForDocumentation/howToDestroyElementsInList.cpp

```
11 TEST(StkMeshHowTo, DestroyElementsInList)
12 {
13     stk::mesh::MetaData metaData;
14     stk::mesh::BulkData bulkData(metaData, MPI_COMM_WORLD);
15     stk::unit_test_util::fill_mesh_using_stk_io("generated:1x1x4", bulkData);
16     EXPECT_GT(stk::mesh::count_selected_entities(metaData.universal_part(),
                bulkData.buckets(stk::topology::ELEM_RANK)), 0u);
17     stk::mesh::EntityVector
                elementsToDestroy{bulkData.get_entity(stk::topology::ELEMENT_RANK,1)};
18     stk::mesh::destroy_elements(bulkData, elementsToDestroy);
19
20     stk::mesh::EntityVector orphanedNodes{
21         bulkData.get_entity(stk::topology::NODE_RANK,1),
22         bulkData.get_entity(stk::topology::NODE_RANK,2),
23         bulkData.get_entity(stk::topology::NODE_RANK,3),
24         bulkData.get_entity(stk::topology::NODE_RANK,4)
25     };
26
27     for(stk::mesh::Entity node : orphanedNodes)
28         EXPECT_FALSE(bulkData.is_valid(node));
29 }
```

1.7 STK Mesh usage examples

This section gives examples of how to access and manipulate a STK Mesh. The examples attempt to give demonstrations of several common tasks that an application developer may want to perform using STK Mesh.

1.7.1 How to iterate over nodes

This example shows how to select the nodes for a subset of the mesh (a surface part), then iterate over those nodes and access the values of a temperature field associated with the nodes.

Listing 1.30: Example of iterating over nodes ../../../code/stk/stk_mesh/testsForDocumentation/howToIterateEntities.cpp

```
55 TEST(StkMeshHowTo, iterateSidesetNodesMostEfficientlyForFieldDataAccess)
56 {
57     MPI_Comm communicator = MPI_COMM_WORLD;
58     if (stk::parallel_machine_size(communicator) != 1) { return; }
59     stk::io::StkMeshIoBroker stkMeshIoBroker(communicator);
60     // syntax creates faces for the surface on the positive 'x-side' of the 2x2x2 cube,
61     // this part is given the name 'surface_1' when it is created [create_input_mesh()]
62     const std::string generatedMeshSpecification = "generated:2x2x2|sideset:X";
63     stkMeshIoBroker.add_mesh_database(generatedMeshSpecification, stk::io::READ_MESH);
64     stkMeshIoBroker.create_input_mesh();
65
66     stk::mesh::MetaData &stkMeshMetaData = stkMeshIoBroker.meta_data();
67     stk::mesh::Field<double> &temperatureField =
               stkMeshMetaData.declare_field<stk::mesh::Field<double>
               >(stk::topology::NODE_RANK, "temperature");
68     stk::mesh::put_field_on_entire_mesh(temperatureField);
69     stkMeshIoBroker.populate_bulk_data();
70
71     stk::mesh::Part &boundaryConditionPart = *stkMeshMetaData.get_part("surface_1");
72     stk::mesh::Selector boundaryNodesSelector(boundaryConditionPart);
73
74     stk::mesh::BulkData &stkMeshBulkData = stkMeshIoBroker.bulk_data();
75     const stk::mesh::BucketVector &boundaryNodeBuckets =
               stkMeshBulkData.get_buckets(stk::topology::NODE_RANK, boundaryNodesSelector);
76
77     double prescribedTemperatureValue = 2.0;
78     std::set<stk::mesh::EntityId> boundaryNodeIds;
79     for (size_t bucketIndex = 0; bucketIndex < boundaryNodeBuckets.size(); ++bucketIndex)
80     {
81         stk::mesh::Bucket &nodeBucket = *boundaryNodeBuckets[bucketIndex];
82         double *temperatureValues = stk::mesh::field_data(temperatureField, nodeBucket);
83         for (size_t nodeIndex = 0; nodeIndex < nodeBucket.size(); ++nodeIndex)
84         {
85             stk::mesh::Entity node = nodeBucket[nodeIndex];
86             boundaryNodeIds.insert(stkMeshBulkData.identifier(node));
87             temperatureValues[nodeIndex] = prescribedTemperatureValue;
88         }
89     }
90
91     testUtils::testTemperatureFieldSetCorrectly(temperatureField, prescribedTemperatureValue,
               boundaryNodeIds);
92 }
93
94 TEST(StkMeshHowTo, iterateSidesetNodesWithFieldDataAccess)
95 {
96     MPI_Comm communicator = MPI_COMM_WORLD;
97     if (stk::parallel_machine_size(communicator) != 1) { return; }
98     stk::io::StkMeshIoBroker stkMeshIoBroker(communicator);
99     // syntax creates faces for the surface on the positive 'x-side' of the 2x2x2 cube,
100    // this part is given the name 'surface_1' when it is created [create_input_mesh()]
101    const std::string generatedMeshSpecification = "generated:2x2x2|sideset:X";
102    stkMeshIoBroker.add_mesh_database(generatedMeshSpecification, stk::io::READ_MESH);
103    stkMeshIoBroker.create_input_mesh();
104
105    stk::mesh::MetaData &stkMeshMetaData = stkMeshIoBroker.meta_data();
106    stk::mesh::Field<double> &temperatureField =
               stkMeshMetaData.declare_field<stk::mesh::Field<double>
               >(stk::topology::NODE_RANK, "temperature");
107    stk::mesh::put_field_on_entire_mesh(temperatureField);
108    stkMeshIoBroker.populate_bulk_data();
109
110    stk::mesh::Part &boundaryConditionPart = *stkMeshMetaData.get_part("surface_1");
111    stk::mesh::Selector boundaryNodesSelector(boundaryConditionPart);
112
113    stk::mesh::BulkData &stkMeshBulkData = stkMeshIoBroker.bulk_data();
114
115    stk::mesh::EntityVector nodes;
```

```
116     stk::mesh::get_selected_entities(boundaryNodesSelector,
                stkMeshBulkData.buckets(stk::topology::NODE_RANK), nodes);
117
118     double prescribedTemperatureValue = 2.0;
119     std::set<stk::mesh::EntityId> boundaryNodeIds;
120
121     for (size_t nodeIndex = 0; nodeIndex < nodes.size(); ++nodeIndex)
122     {
123         boundaryNodeIds.insert(stkMeshBulkData.identifier(nodes[nodeIndex]));
124         double *temperatureValues = stk::mesh::field_data(temperatureField, nodes[nodeIndex]);
125         *temperatureValues = prescribedTemperatureValue;
126     }
127
128     testUtils::testTemperatureFieldSetCorrectly(temperatureField, prescribedTemperatureValue,
                boundaryNodeIds);
129 }
```

1.7.2 How to traverse connectivity

`stk::mesh::BulkData` provides member functions for accessing connectivity data by entity and rank. The implementations of these `BulkData` methods must first look up the bucket for the given entity and rank and the entity's index in that bucket. When iterating through the entities in a given bucket, it is therefore more efficient to access this connectivity data through a second connectivity API that STK Mesh provides on the Bucket.

Listing 1.31: Example of how to traverse connectivity via accessors on BulkData and via accessors on Bucket ../../../code/stk/stk_mesh/testsForDocumentation/howToIterateConnectivity.cpp

```
54 TEST(StkMeshHowTo, iterateConnectivityThroughBulkData)
55 {
56     MPI_Comm communicator = MPI_COMM_WORLD;
57     if (stk::parallel_machine_size(communicator) != 1) { return; }
58     stk::io::StkMeshIoBroker stkMeshIoBroker(communicator);
59     // Generate a mesh of hexes with a sideset
60     const std::string generatedMeshSpecification = "generated:2x2x2|sideset:X";
61     stkMeshIoBroker.add_mesh_database(generatedMeshSpecification, stk::io::READ_MESH);
62     stkMeshIoBroker.create_input_mesh();
63     stkMeshIoBroker.populate_bulk_data();
64
65     stk::mesh::MetaData &stkMeshMetaData = stkMeshIoBroker.meta_data();
66     stk::mesh::BulkData &stkMeshBulkData = stkMeshIoBroker.bulk_data();
67     const stk::mesh::BucketVector &elementBuckets =
68         stkMeshBulkData.buckets(stk::topology::ELEMENT_RANK);
69
70     typedef stk::mesh::Field<double, stk::mesh::Cartesian> CoordinatesField_t;
71     CoordinatesField_t const & coord_field =
72           *dynamic_cast<CoordinatesField_t const *>(stkMeshMetaData.coordinate_field());
73
74     const unsigned nodesPerHex = 8;
75     const unsigned spatialDim = 3;
76     unsigned count = 0;
77     double elementNodeCoords[nodesPerHex][spatialDim];
78     for (size_t bucketIndex = 0; bucketIndex < elementBuckets.size(); ++bucketIndex)
79     {
80         stk::mesh::Bucket &elemBucket = *elementBuckets[bucketIndex];
81         for (size_t elemIndex = 0; elemIndex < elemBucket.size(); ++elemIndex)
82         {
83             stk::mesh::Entity elem = elemBucket[elemIndex];
84             unsigned numNodes = stkMeshBulkData.num_nodes(elem);
85             EXPECT_EQ(numNodes, nodesPerHex);
```

```
            stk::mesh::Entity const* nodes = stkMeshBulkData.begin_nodes(elem);
            for (unsigned inode = 0; inode < numNodes; ++inode)
            {
              double *coords = stk::mesh::field_data(coord_field, nodes[inode]);
              elementNodeCoords[inode][0] = coords[0];
              elementNodeCoords[inode][1] = coords[1];
              elementNodeCoords[inode][2] = coords[2];
              EXPECT_NE(elementNodeCoords[inode][0], std::numeric_limits<double>::max());
              EXPECT_NE(elementNodeCoords[inode][1], std::numeric_limits<double>::max());
              EXPECT_NE(elementNodeCoords[inode][2], std::numeric_limits<double>::max());
              ++count;
            }
        }
    }
    EXPECT_GE(count, 1u);
}

TEST(StkMeshHowTo, iterateConnectivityThroughBuckets)
{
    MPI_Comm communicator = MPI_COMM_WORLD;
    if (stk::parallel_machine_size(communicator) != 1) { return; }
    stk::io::StkMeshIoBroker stkMeshIoBroker(communicator);
    // Generate a mesh of hexes with a sideset
    const std::string generatedMeshSpecification = "generated:2x2x2|sideset:X";
    stkMeshIoBroker.add_mesh_database(generatedMeshSpecification, stk::io::READ_MESH);
    stkMeshIoBroker.create_input_mesh();
    stkMeshIoBroker.populate_bulk_data();

    stk::mesh::MetaData &stkMeshMetaData = stkMeshIoBroker.meta_data();
    stk::mesh::BulkData &stkMeshBulkData = stkMeshIoBroker.bulk_data();
    const stk::mesh::BucketVector &elementBuckets =
        stkMeshBulkData.buckets(stk::topology::ELEMENT_RANK);

    typedef stk::mesh::Field<double, stk::mesh::Cartesian> CoordinatesField_t;
    CoordinatesField_t const & coord_field =
          *dynamic_cast<CoordinatesField_t const *>(stkMeshMetaData.coordinate_field());

    const unsigned nodesPerHex = 8;
    const unsigned spatialDim = 3;
    unsigned count = 0;
    double elementNodeCoords[nodesPerHex][spatialDim];
    for (size_t bucketIndex = 0; bucketIndex < elementBuckets.size(); ++bucketIndex)
    {
        stk::mesh::Bucket &elemBucket = *elementBuckets[bucketIndex];
        for (size_t elemIndex = 0; elemIndex < elemBucket.size(); ++elemIndex)
        {
            unsigned numNodes = elemBucket.num_nodes(elemIndex);
            EXPECT_EQ(numNodes, nodesPerHex);
            stk::mesh::Entity const* nodes = elemBucket.begin_nodes(elemIndex);
            for (unsigned inode = 0; inode < numNodes; ++inode)
            {
              double *coords = stk::mesh::field_data(coord_field, nodes[inode]);
              elementNodeCoords[inode][0] = coords[0];
              elementNodeCoords[inode][1] = coords[1];
              elementNodeCoords[inode][2] = coords[2];
              EXPECT_NE(elementNodeCoords[inode][0], std::numeric_limits<double>::max());
              EXPECT_NE(elementNodeCoords[inode][1], std::numeric_limits<double>::max());
              EXPECT_NE(elementNodeCoords[inode][2], std::numeric_limits<double>::max());
              ++count;
            }
        }
    }
    EXPECT_GE(count, 1u);
}
```

1.7.3 How to check side equivalency

Listing 1.32: Example of how to check side equivalency
../../../code/stk/stk_mesh/testsForDocumentation/howToUseEquivalent.cpp

```
19 TEST_F(MeshWithSide, whenCheckingSideEquivalency_returnsCorrectPermutation)
20 {
21     if (stk::parallel_machine_size(get_comm()) == 1) {
22         setup_mesh("generated:1x1x4|sideset:x", stk::mesh::BulkData::NO_AUTO_AURA);
23         stk::mesh::Entity elem1 = get_bulk().get_entity(stk::topology::ELEM_RANK, 1);
24         ASSERT_EQ(1u, get_bulk().num_faces(elem1));
25         const stk::mesh::Entity side = *get_bulk().begin_faces(elem1);
26         const stk::mesh::Permutation perm = *get_bulk().begin_face_permutations(elem1);
27         const stk::mesh::ConnectivityOrdinal ordinal = *get_bulk().begin_face_ordinals(elem1);
28         const stk::mesh::Entity* sideNodes = get_bulk().begin_nodes(side);
29
30         std::pair<bool,unsigned> equivAndPermutation = stk::mesh::side_equivalent(get_bulk(),
               elem1, ordinal, sideNodes);
31         EXPECT_TRUE(equivAndPermutation.first);
32         EXPECT_EQ(perm, static_cast<stk::mesh::Permutation>(equivAndPermutation.second));
33
34         EXPECT_TRUE(stk::mesh::is_side_equivalent(get_bulk(), elem1, ordinal, sideNodes));
35
36         stk::mesh::EquivAndPositive result =
               stk::mesh::is_side_equivalent_and_positive(get_bulk(), elem1, ordinal,
               sideNodes);
37         EXPECT_TRUE(result.is_equiv);
38         EXPECT_TRUE(result.is_positive);
39     }
40 }
```

1.7.4 Understanding node ordering of edges and faces

Listing 1.33 shows the difference between node orderings when using the STK Mesh `create_edges()` and `create_faces()` functions versus STK Topology. Listing 2.10 has more information regarding the lexicographical smallest permutation which is used to change the ordering for the two cases.

Listing 1.33: Understanding edge and face ordering
../../../code/stk/stk_mesh/testsForDocumentation/createFacesEdgesHex.cpp

```
216     // ============================================================
217     //+ EXAMPLE
218     //+ Create the faces..
219     stk::mesh::create_faces(bulkData);
220
221     unsigned goldValuesForHexFaceNodesFromStkTopology[6][4] = {
222         {1, 2, 6, 5}, {2, 3, 7, 6}, {3, 4, 8, 7}, {1, 5, 8, 4}, {1, 4, 3, 2}, {5, 6, 7, 8} };
223
224     // Lexicographical smallest permutation per face leads from topology ordering (above) for
            face to ordering below
225
226     unsigned goldValuesForHexFaceNodesFromCreateFaces[6][4] = {
227         {1, 2, 6, 5}, {2, 3, 7, 6}, {3, 4, 8, 7}, {1, 4, 8, 5}, {1, 2, 3, 4}, {5, 6, 7, 8} };
228
229     //+ Create the edges..
230     stk::mesh::create_edges(bulkData);
231
232     unsigned goldValuesHexEdgeNodesFromStkTopology[12][2] = {
233       {1, 2}, {2, 3}, {3, 4}, {4, 1}, {5, 6}, {6, 7}, {7, 8}, {8, 5}, {1, 5}, {2, 6}, {3, 7},
            {4, 8} };
```

```
234
235     // Lexicographical smallest permutation per edge leads from topology ordering (above) for
                edge to ordering below
236
237     unsigned goldValuesHexEdgeNodesFromCreateEdges[12][2] = {
238       {1, 2}, {2, 3}, {3, 4}, {1, 4}, {5, 6}, {6, 7}, {7, 8}, {5, 8}, {1, 5}, {2, 6}, {3, 7},
                {4, 8} };
239
240
```

1.7.5 How to sort entities into an arbitrary order

One possible use case for this is to try and improve cache hit rate when visiting the nodes of an element.

Listing 1.34: Example showing how to sort entities by descending identifier. ../../../code/stk/stk_mesh/testsForDocumentation/howToSortEntities.cpp

```
1  #include "gtest/gtest.h"
2  #include <stk_mesh/base/BulkData.hpp>
3  #include <stk_unit_test_utils/MeshFixture.hpp>
4
5  namespace {
6
7  class EntityReverseSorter : public stk::mesh::EntitySorterBase
8  {
9  public:
10     virtual void sort(stk::mesh::BulkData &bulk, stk::mesh::EntityVector& entityVector) const
11     {
12         std::sort(entityVector.begin(), entityVector.end(),
13                   [&bulk](stk::mesh::Entity a, stk::mesh::Entity b) { return
                      bulk.identifier(a) > bulk.identifier(b); });
14     }
15 };
16
17 class HowToSortEntities : public stk::unit_test_util::MeshFixture
18 {
19 protected:
20     void sort_and_check()
21     {
22         if(stk::parallel_machine_size(get_comm()) == 1)
23         {
24             setup_mesh("generated:1x1x4", stk::mesh::BulkData::AUTO_AURA);
25             get_bulk().sort_entities(EntityReverseSorter());
26             expect_entities_in_reverse_order();
27         }
28     }
29     void expect_entities_in_reverse_order()
30     {
31         const stk::mesh::BucketVector buckets = get_bulk().buckets(stk::topology::NODE_RANK);
32         ASSERT_EQ(1u, buckets.size());
33         expect_bucket_in_reverse_order(*buckets[0]);
34     }
35     void expect_bucket_in_reverse_order(const stk::mesh::Bucket &bucket)
36     {
37         ASSERT_EQ(20u, bucket.size());
38         for(size_t i=1; i<bucket.size(); i++)
39             EXPECT_GT(get_bulk().identifier(bucket[i-1]), get_bulk().identifier(bucket[i]));
40     }
41 };
42 TEST_F(HowToSortEntities, example_reverse)
```

```
43 {
44     sort_and_check();
45 }
46
47 }
```

Chapter 2

STK Topology

As stated in the introductory chapter, *Topology* provides an entity's finite element description and this includes a number of attributes such as the number and type of lower-rank entities that can exist in that entity's downward connectivity (e.g., the number of faces that an element topology can have, the ordering of nodes attached to particular faces, etc.).

A primary goal of `stk_topology` is to provide fast traversal of sub-topologies, such as the edges of an element or the nodes of a face, etc. `stk_topology` uses value semantics (e.g., no pointers to singletons) and can be used on GPUs as well as CPUs. `stk_topology` provides compile-time access to topology information, as well as run-time. (See Section 2.1.3, Listing 2.3).

2.1 STK Topology API

This section contains several code listings that attempt to aid in the understanding of the stk topology API.

Note the following details of the API:

- `num_nodes()` vs `num_vertices()`: For linear topologies, the number of nodes equals the number of vertices. For higher order topologies, "nodes" include those located at the corners as well as those located at mide-sides and/or mid-edges; but "vertices" are only those nodes located at the corners.
- `is_shell()`: This is a helper to distinguish between "structural" elements (such as shells and beams), and "continuum" elements.
- Permutations (`num_permutations()` vs `num_positive_permutations()`): Different orderings of a topology's nodes may appear in certain contexts. Positive vs negative refers to whether a given node ordering represents a different direction "normal" for that topology. Note also that this isn't a true mathematical permutation since not all possible "permutations" of the nodes are even valid; these permutations are essentially node traversals with the same sequence but different starting points.
- `base()`: For topologies with polynomial order higher than linear, "base()" provides the corresponding linear topology.

- `is_superelement()`, `create_superelement_topology()`: Super-elements are used for reduced-order modeling in certain application formulations.

2.1.1 How to set and get topology

This example shows how to attach topology to entities (if entities are created "in line" rather than being created by STK IO). Essentially, topology is attached to entities by declaring the entities to be members of a Part that has the desired topology. The example also shows how to retrieve topology from the mesh. More detailed information about STK Topology is provided in Chapter 2.

Listing 2.1: Example of setting/getting topology
../../../code/stk/stk_mesh/testsForDocumentation/setAndGetTopology.cpp

```
61 TEST(stkMeshHowTo, setAndGetTopology)
62 {
63     const unsigned spatialDimension = 3;
64     stk::mesh::MetaData metaData(spatialDimension, stk::mesh::entity_rank_names());
65     stk::mesh::Part &tetPart = metaData.declare_part_with_topology("tet part",
           stk::topology::TET_4);
66
67     stk::mesh::Part &hexPart = metaData.declare_part("existing part with currently unknown
           topology");
68     // . . . then later assigned
69     stk::mesh::set_topology(hexPart, stk::topology::HEX_8);
70
71     metaData.commit();
72     stk::mesh::BulkData bulkData(metaData, MPI_COMM_WORLD);
73
74     bulkData.modification_begin();
75     stk::mesh::EntityId elem1Id = 1, elem2Id = 2;
76     stk::mesh::Entity elem1 = bulkData.declare_entity(stk::topology::ELEMENT_RANK, elem1Id,
           tetPart);
77     stk::mesh::Entity elem2 = bulkData.declare_entity(stk::topology::ELEMENT_RANK, elem2Id,
           hexPart);
78     declare_element_nodes(bulkData, elem1, elem2);
79     bulkData.modification_end();
80
81     stk::topology elem1_topology = bulkData.bucket(elem1).topology();
82     stk::topology elem2_topology = bulkData.bucket(elem2).topology();
83
84     EXPECT_EQ(stk::topology::TET_4, elem1_topology);
85     EXPECT_EQ(stk::topology::HEX_8, elem2_topology);
86 }
```

2.1.2 STK topology ranks

Listing 2.2 demonstrates the link between various STK topologies and their ranks.

Listing 2.2: Example showing mapping of STK topologies to ranks
../../../code/stk/stk_topology/tests_for_documentation/map_stk_topologies_to_ranks.cpp

```
41 TEST(stk_topology_how_to, map_topologies_to_ranks )
42 {
43     stk::topology topology = stk::topology::INVALID_TOPOLOGY;
44     EXPECT_EQ(stk::topology::INVALID_RANK, topology.rank());
```

```
45
46      std::vector<stk::topology> node_rank_topologies;
47      node_rank_topologies.push_back(stk::topology::NODE);
48
49      std::vector<stk::topology> edge_rank_topologies;
50      edge_rank_topologies.push_back(stk::topology::LINE_2);
51      edge_rank_topologies.push_back(stk::topology::LINE_3);
52
53      std::vector<stk::topology> face_rank_topologies;
54      face_rank_topologies.push_back(stk::topology::TRI_3);
55      face_rank_topologies.push_back(stk::topology::TRIANGLE_3);
56      face_rank_topologies.push_back(stk::topology::TRI_4);
57      face_rank_topologies.push_back(stk::topology::TRIANGLE_4);
58      face_rank_topologies.push_back(stk::topology::TRI_6);
59      face_rank_topologies.push_back(stk::topology::TRIANGLE_6);
60      face_rank_topologies.push_back(stk::topology::QUAD_4);
61      face_rank_topologies.push_back(stk::topology::QUADRILATERAL_4);
62      face_rank_topologies.push_back(stk::topology::QUAD_8);
63      face_rank_topologies.push_back(stk::topology::QUADRILATERAL_8);
64      face_rank_topologies.push_back(stk::topology::QUAD_9);
65      face_rank_topologies.push_back(stk::topology::QUADRILATERAL_9);
66
67      std::vector<stk::topology> element_rank_topologies;
68      element_rank_topologies.push_back(stk::topology::PARTICLE);
69      element_rank_topologies.push_back(stk::topology::LINE_2_1D);
70      element_rank_topologies.push_back(stk::topology::LINE_3_1D);
71      element_rank_topologies.push_back(stk::topology::BEAM_2);
72      element_rank_topologies.push_back(stk::topology::BEAM_3);
73      element_rank_topologies.push_back(stk::topology::SHELL_LINE_2);
74      element_rank_topologies.push_back(stk::topology::SHELL_LINE_3);
75
76      element_rank_topologies.push_back(stk::topology::TRI_3_2D);
77      element_rank_topologies.push_back(stk::topology::TRIANGLE_3_2D);
78      element_rank_topologies.push_back(stk::topology::TRI_4_2D);
79      element_rank_topologies.push_back(stk::topology::TRIANGLE_4_2D);
80      element_rank_topologies.push_back(stk::topology::TRI_6_2D);
81      element_rank_topologies.push_back(stk::topology::TRIANGLE_6_2D);
82      element_rank_topologies.push_back(stk::topology::QUAD_4_2D);
83      element_rank_topologies.push_back(stk::topology::QUADRILATERAL_4_2D);
84      element_rank_topologies.push_back(stk::topology::QUAD_8_2D);
85      element_rank_topologies.push_back(stk::topology::QUADRILATERAL_8_2D);
86      element_rank_topologies.push_back(stk::topology::QUAD_9_2D);
87      element_rank_topologies.push_back(stk::topology::QUADRILATERAL_9_2D);
88
89      element_rank_topologies.push_back(stk::topology::SHELL_TRI_3);
90      element_rank_topologies.push_back(stk::topology::SHELL_TRIANGLE_3);
91      element_rank_topologies.push_back(stk::topology::SHELL_TRI_4);
92      element_rank_topologies.push_back(stk::topology::SHELL_TRIANGLE_4);
93      element_rank_topologies.push_back(stk::topology::SHELL_TRI_6);
94      element_rank_topologies.push_back(stk::topology::SHELL_TRIANGLE_6);
95
96      element_rank_topologies.push_back(stk::topology::SHELL_QUAD_4);
97      element_rank_topologies.push_back(stk::topology::SHELL_QUADRILATERAL_4);
98      element_rank_topologies.push_back(stk::topology::SHELL_QUAD_8);
99      element_rank_topologies.push_back(stk::topology::SHELL_QUADRILATERAL_8);
100     element_rank_topologies.push_back(stk::topology::SHELL_QUAD_9);
101     element_rank_topologies.push_back(stk::topology::SHELL_QUADRILATERAL_9);
102
103     element_rank_topologies.push_back(stk::topology::TET_4);
104     element_rank_topologies.push_back(stk::topology::TETRAHEDRON_4);
105     element_rank_topologies.push_back(stk::topology::TET_8);
106     element_rank_topologies.push_back(stk::topology::TETRAHEDRON_8);
107     element_rank_topologies.push_back(stk::topology::TET_10);
108     element_rank_topologies.push_back(stk::topology::TETRAHEDRON_10);
109     element_rank_topologies.push_back(stk::topology::TET_11);
110     element_rank_topologies.push_back(stk::topology::TETRAHEDRON_11);
111
112     element_rank_topologies.push_back(stk::topology::PYRAMID_5);
```

```
113     element_rank_topologies.push_back(stk::topology::PYRAMID_13);
114     element_rank_topologies.push_back(stk::topology::PYRAMID_14);
115     element_rank_topologies.push_back(stk::topology::WEDGE_6);
116     element_rank_topologies.push_back(stk::topology::WEDGE_15);
117     element_rank_topologies.push_back(stk::topology::WEDGE_18);
118     element_rank_topologies.push_back(stk::topology::QUADRILATERAL_9_2D);
119     element_rank_topologies.push_back(stk::topology::QUADRILATERAL_9_2D);
120
121     element_rank_topologies.push_back(stk::topology::HEX_8);
122     element_rank_topologies.push_back(stk::topology::HEXAHEDRON_8);
123     element_rank_topologies.push_back(stk::topology::HEX_20);
124     element_rank_topologies.push_back(stk::topology::HEXAHEDRON_20);
125     element_rank_topologies.push_back(stk::topology::HEX_27);
126     element_rank_topologies.push_back(stk::topology::HEXAHEDRON_27);
127
128     unsigned num_nodes_in_super_element = 10;
129     element_rank_topologies.
130       push_back(stk::create_superelement_topology(num_nodes_in_super_element));
```

2.1.3 Compile-time STK topology information

Listing 2.3 demonstrates how to access compile-time topology information. In this example, `compiletime_num_nodes` is a variable that is assigned a constant, compile-time value. `compiletime_hex8` is a type of **`struct`**, and `num_nodes` is a static const member whose value is defined at compile-time. It thus can be used to allocate space on the stack instead of on the heap. Other compile-time topology attributes are defined by the members of the `topology::topology_type` struct in the file `stk_topology/topology_type.tcc`.

Listing 2.3: Example using compile-time STK topology information ../../../code/stk/stk_topology/tests_for_documentation/runtime_vs_compiletime_topology.cpp

```
39 TEST(stk_topology_how_to, runtime_vs_compiletime_topology )
40 {
41     stk::topology runtime_hex8 = stk::topology::HEX_8;
42
43     typedef stk::topology::topology_type<stk::topology::HEX_8> compiletime_hex8;
44
45     const unsigned compiletime_num_nodes = compiletime_hex8::num_nodes;
46
47     EXPECT_EQ( runtime_hex8.num_nodes(), compiletime_num_nodes );
48
49     //declare a static array with length given by compile-time num-nodes
50     double compile_time_sized_array[compiletime_num_nodes];
51     EXPECT_EQ(sizeof(compile_time_sized_array), sizeof(double)*compiletime_num_nodes);
52 }
```

2.1.4 STK topology for the Particle

Listing 2.4 demonstrates the API for a Particle element.

Listing 2.4: Example showing STK topology for a zero-dimensional element ../../../code/stk/stk_topology/tests_for_documentation/element_topologies.cpp

```
41 TEST(stk_topology_understanding, zero_dim_element)
42 {
43     stk::topology sphere = stk::topology::PARTICLE;
44
45     EXPECT_TRUE(sphere.is_valid());
46     EXPECT_FALSE(sphere.has_homogeneous_faces());
47     EXPECT_FALSE(sphere.is_shell());
48
49     EXPECT_TRUE(sphere.rank() != stk::topology::NODE_RANK);
50     EXPECT_TRUE(sphere.rank() != stk::topology::EDGE_RANK);
51     EXPECT_TRUE(sphere.rank() != stk::topology::FACE_RANK);
52     EXPECT_TRUE(sphere.rank() != stk::topology::CONSTRAINT_RANK);
53     EXPECT_TRUE(sphere.rank() == stk::topology::ELEMENT_RANK);
54
55     EXPECT_EQ(sphere.side_rank(), stk::topology::INVALID_RANK);
56
57     EXPECT_EQ(sphere.dimension(),1u);
58     EXPECT_EQ(sphere.num_nodes(),1u);
59     EXPECT_EQ(sphere.num_vertices(),1u);
60     EXPECT_EQ(sphere.num_edges(),0u);
61     EXPECT_EQ(sphere.num_faces(),0u);
62     EXPECT_EQ(sphere.num_sides(),0u);
63     EXPECT_EQ(sphere.num_permutations(),1u);
64     EXPECT_EQ(sphere.num_positive_permutations(),1u);
65
66     EXPECT_FALSE(sphere.defined_on_spatial_dimension(0));
67
68     EXPECT_TRUE(sphere.defined_on_spatial_dimension(1));
69     EXPECT_TRUE(sphere.defined_on_spatial_dimension(2));
70     EXPECT_TRUE(sphere.defined_on_spatial_dimension(3));
71
72     EXPECT_EQ(sphere.base(),stk::topology::PARTICLE);
73 }
```

2.1.5 STK topology for the high order Beam

Listing 2.5 demonstrates the API for a higher order Beam element.

Listing 2.5: Example of STK topology for a one-dimensional element ../../../code/stk/stk_topology/tests_for_documentation/element_topologies.cpp

```
157 TEST(stk_topology_understanding, one_dim_higher_order_element)
158 {
159     stk::topology secondOrderBeam = stk::topology::BEAM_3;
160
161     EXPECT_TRUE(secondOrderBeam.is_valid());
162     EXPECT_FALSE(secondOrderBeam.has_homogeneous_faces());
163     EXPECT_FALSE(secondOrderBeam.is_shell());
164
165     EXPECT_TRUE(secondOrderBeam.rank() != stk::topology::NODE_RANK);
166     EXPECT_TRUE(secondOrderBeam.rank() != stk::topology::EDGE_RANK);
167     EXPECT_TRUE(secondOrderBeam.rank() != stk::topology::FACE_RANK);
168     EXPECT_TRUE(secondOrderBeam.rank() != stk::topology::CONSTRAINT_RANK);
169     EXPECT_TRUE(secondOrderBeam.rank() == stk::topology::ELEMENT_RANK);
170
171     EXPECT_TRUE(secondOrderBeam.side_rank() == stk::topology::EDGE_RANK);
172
173     EXPECT_EQ(2u, secondOrderBeam.dimension());
174     EXPECT_EQ(3u, secondOrderBeam.num_nodes());
175     EXPECT_EQ(2u, secondOrderBeam.num_vertices());
176     EXPECT_EQ(1u, secondOrderBeam.num_edges());
177
```

```
178     EXPECT_EQ(0u, secondOrderBeam.num_faces());
179     EXPECT_EQ(1u, secondOrderBeam.num_positive_permutations());
180     EXPECT_EQ(2u, secondOrderBeam.num_permutations());
181
182     EXPECT_FALSE(secondOrderBeam.defined_on_spatial_dimension(0));
183     EXPECT_FALSE(secondOrderBeam.defined_on_spatial_dimension(1));
184
185     EXPECT_TRUE(secondOrderBeam.defined_on_spatial_dimension(2));
186     EXPECT_TRUE(secondOrderBeam.defined_on_spatial_dimension(3));
187
188     EXPECT_TRUE(secondOrderBeam.base() == stk::topology::BEAM_2);
189
190     unsigned beamNodes[3] = { 10, 20, 14 }; // 10 *-------*-------* 20
191                                             //            14
192     {
193         unsigned expectedNodeOffsets[3] = { 0, 1, 2 };
194         //unit-test checking utility:
195         checkNodeOrderingAndOffsetsForEdges(secondOrderBeam, beamNodes, expectedNodeOffsets);
196     }
197
198     {
199         unsigned expectedNodeOffsets[6] = {
200                 0, 1, 2,
201                 1, 0, 2
202         };
203
204         //unit-test checking utility:
205         checkNodeOrderingAndOffsetsForPermutations(secondOrderBeam, beamNodes,
                expectedNodeOffsets);
206     }
207 }
```

2.1.6 STK topology for the high order triangular Shell

Listing 2.6 demonstrates the API for a higher order triangular shell element.

Listing 2.6: Example of STK topology for a two-dimensional element ../../../code/stk/stk_topology/tests_for_documentation/element_topologies.cpp

```
210 TEST(stk_topology_understanding, two_dim_higher_order_element)
211 {
212     stk::topology secondOrderTriShell = stk::topology::SHELL_TRIANGLE_6;
213     EXPECT_TRUE(secondOrderTriShell == stk::topology::SHELL_TRI_6);
214
215     EXPECT_TRUE(secondOrderTriShell.is_valid());
216     EXPECT_TRUE(secondOrderTriShell.has_homogeneous_faces());
217     EXPECT_TRUE(secondOrderTriShell.is_shell());
218
219     EXPECT_TRUE(secondOrderTriShell.rank() != stk::topology::NODE_RANK);
220     EXPECT_TRUE(secondOrderTriShell.rank() != stk::topology::EDGE_RANK);
221     EXPECT_TRUE(secondOrderTriShell.rank() != stk::topology::FACE_RANK);
222     EXPECT_TRUE(secondOrderTriShell.rank() != stk::topology::CONSTRAINT_RANK);
223     EXPECT_TRUE(secondOrderTriShell.rank() == stk::topology::ELEMENT_RANK);
224
225     EXPECT_TRUE(secondOrderTriShell.side_rank() == stk::topology::FACE_RANK);
226
227     EXPECT_EQ(3u, secondOrderTriShell.dimension());
228     EXPECT_EQ(6u, secondOrderTriShell.num_nodes());
229     EXPECT_EQ(3u, secondOrderTriShell.num_vertices());
230     EXPECT_EQ(3u, secondOrderTriShell.num_edges());
231     EXPECT_EQ(2u, secondOrderTriShell.num_faces());
232
```

```
233     // permutations are the number of ways the number of vertices can be permuted
234     EXPECT_EQ(6u, secondOrderTriShell.num_permutations());
235     // positive permutations are ones that the normal is maintained
236     EXPECT_EQ(3u, secondOrderTriShell.num_positive_permutations());
237
238     EXPECT_FALSE(secondOrderTriShell.defined_on_spatial_dimension(0));
239     EXPECT_FALSE(secondOrderTriShell.defined_on_spatial_dimension(1));
240     EXPECT_FALSE(secondOrderTriShell.defined_on_spatial_dimension(2));
241
242     EXPECT_TRUE(secondOrderTriShell.defined_on_spatial_dimension(3));
243
244     EXPECT_TRUE(secondOrderTriShell.base() == stk::topology::SHELL_TRI_3);
245     EXPECT_TRUE(secondOrderTriShell.base() == stk::topology::SHELL_TRIANGLE_3);
246
247     unsigned shellNodes[6] = { 10, 11, 12, 100, 101, 102 }; // first 3 are vertex nodes
            (picture?)
248
249     {
250         unsigned goldValuesEdgeOffsets[9] = {
251                 0, 1, 3,
252                 1, 2, 4,
253                 2, 0, 5
254         };
255
256         //unit-test checking utility:
257         checkNodeOrderingAndOffsetsForEdges(secondOrderTriShell, shellNodes,
                goldValuesEdgeOffsets);
258     }
259
260     {
261         unsigned goldValuesFaceNodeOffsets[12] = {
262                 0, 1, 2, 3, 4, 5,
263                 0, 2, 1, 5, 4, 3
264         };
265
266         //unit-test checking utility:
267         checkNodeOrderingAndOffsetsForFaces(secondOrderTriShell, shellNodes,
                goldValuesFaceNodeOffsets);
268     }
269
270     {
271         unsigned goldValueOffsetsPerm[36] = {
272                 0, 1, 2, 3, 4, 5,
273                 2, 0, 1, 5, 3, 4,
274                 1, 2, 0, 4, 5, 3,
275                 0, 2, 1, 5, 4, 3,
276                 2, 1, 0, 4, 3, 5,
277                 1, 0, 2, 3, 5, 4
278         };
279
280         //unit-test checking utility:
281         checkNodeOrderingAndOffsetsForPermutations(secondOrderTriShell, shellNodes,
                goldValueOffsetsPerm);
282     }
283 }
```

2.1.7 STK topology for the linear Hexahedral

Listing 2.7 demonstrates the API for a linear Hexahedral element.

Listing 2.7: Example of STK topology for a three-dimensional element ../../../code/stk/stk_topology/tests_for_documentation/element_topologies.cpp

```
TEST(stk_topology_understanding, three_dim_linear_element)
{
    stk::topology hex8 = stk::topology::HEX_8;
    EXPECT_TRUE(hex8 == stk::topology::HEXAHEDRON_8);

    EXPECT_TRUE(hex8.is_valid());
    EXPECT_TRUE(hex8.has_homogeneous_faces());
    EXPECT_FALSE(hex8.is_shell());

    EXPECT_TRUE(hex8.rank() != stk::topology::NODE_RANK);
    EXPECT_TRUE(hex8.rank() != stk::topology::EDGE_RANK);
    EXPECT_TRUE(hex8.rank() != stk::topology::FACE_RANK);
    EXPECT_TRUE(hex8.rank() != stk::topology::CONSTRAINT_RANK);
    EXPECT_TRUE(hex8.rank() == stk::topology::ELEMENT_RANK);

    EXPECT_TRUE(hex8.side_rank() == stk::topology::FACE_RANK);

    EXPECT_EQ(3u, hex8.dimension());
    EXPECT_EQ(8u, hex8.num_nodes());
    EXPECT_EQ(8u, hex8.num_vertices());
    EXPECT_EQ(12u, hex8.num_edges());
    EXPECT_EQ(6u, hex8.num_faces());

    if (stk::topology::topology_type<stk::topology::HEX_8>::num_permutations > 1) {
        // permutations are the number of ways the number of vertices can be permuted
        EXPECT_EQ(24u, hex8.num_permutations());
        // positive permutations are ones that the normal is maintained
        EXPECT_EQ(24u, hex8.num_positive_permutations());
    }

    EXPECT_FALSE(hex8.defined_on_spatial_dimension(0));
    EXPECT_FALSE(hex8.defined_on_spatial_dimension(1));
    EXPECT_FALSE(hex8.defined_on_spatial_dimension(2));

    EXPECT_TRUE(hex8.defined_on_spatial_dimension(3));

    EXPECT_TRUE(hex8.base() == stk::topology::HEX_8);

    unsigned hex8Nodes[8] = { 0, 1, 2, 3, 4, 5, 6, 7 };

    {
        stk::topology goldEdgeTopology = stk::topology::LINE_2;
        EXPECT_EQ(goldEdgeTopology, hex8.edge_topology());

        unsigned goldNumNodesPerEdge = 2;
        ASSERT_EQ(goldNumNodesPerEdge, hex8.edge_topology().num_nodes());
        unsigned goldValuesEdgeOffsets[24] = {
                0, 1,
                1, 2,
                2, 3,
                3, 0,
                4, 5,
                5, 6,
                6, 7,
                7, 4,
                0, 4,
                1, 5,
                2, 6,
                3, 7
        };

        //unit-test checking utility:
        checkNodeOrderingAndOffsetsForEdges(hex8, hex8Nodes, goldValuesEdgeOffsets);
    }

    {
        stk::topology goldFaceTopology = stk::topology::QUAD_4;
        unsigned goldNumNodesPerFace = 4;
```

```
355         for (unsigned faceIndex=0;faceIndex<hex8.num_faces();faceIndex++)
356         {
357             EXPECT_EQ(goldFaceTopology, hex8.face_topology(faceIndex));
358             ASSERT_EQ(goldNumNodesPerFace, hex8.face_topology(faceIndex).num_nodes());
359         }
360 
361         unsigned goldValuesFaceOffsets[24] = {
362                 0, 1, 5, 4,
363                 1, 2, 6, 5,
364                 2, 3, 7, 6,
365                 0, 4, 7, 3,
366                 0, 3, 2, 1,
367                 4, 5, 6, 7
368         };
369 
370         //unit-test checking utility:
371         checkNodeOrderingAndOffsetsForFaces(hex8, hex8Nodes, goldValuesFaceOffsets);
372     }
373 
374 }
```

2.1.8 STK topology `equivalent` method

Listing 2.8 demonstrates the API for checking, given the nodes of topology, if two entities are equivalent. *The support for HEX_8, etc., only includes positive node-permutations, since there is no current need for negative permutations.*

Listing 2.8: Example using of an `equivalent` method ../../../code/stk/stk_topology/tests_for_documentation/equivalent.cpp

```
41 TEST(stk_topology_understanding, equivalent_elements)
42 {
43     std::pair<bool, unsigned> areElementsEquivalent;
44 
45     {
46         if (stk::topology::topology_type<stk::topology::HEX_8>::num_permutations > 1) {
47             unsigned hex1[8] = { 0, 1, 2, 3, 4, 5, 6, 7 };
48             unsigned hex2[8] = { 4, 7, 6, 5, 0, 3, 2, 1 };
49             unsigned hex3[8] = { 4, 5, 6, 7, 0, 1, 2, 3 };
50 
51             stk::topology hex8 = stk::topology::HEX_8;
52 
53             areElementsEquivalent = hex8.equivalent(hex1, hex2);
54             EXPECT_TRUE(areElementsEquivalent.first);
55             areElementsEquivalent = hex8.equivalent(hex1, hex3);
56             EXPECT_FALSE(areElementsEquivalent.first);
57         }
58     }
59 
60     {
61         unsigned triangle_1[3] = {0, 1, 2};
62         unsigned triangle_2[3] = {0, 2, 1};
63 
64         stk::topology triangular_shell = stk::topology::SHELL_TRIANGLE_3;
65 
66         areElementsEquivalent = triangular_shell.equivalent(triangle_1, triangle_2);
67 
68         EXPECT_TRUE(areElementsEquivalent.first);
69 
70         unsigned permutation_index = areElementsEquivalent.second;
71         unsigned goldValue = 3;
```

```
72          EXPECT_EQ(goldValue, permutation_index); // From previous unit test, this is the 4th
                permutation
73      }
74
75  }
```

2.1.9 STK topology's `is_positive_polarity` method

Listing 2.9: Example using `is_positive_polarity`
../../../code/stk/stk_topology/tests_for_documentation/how_to_use_stk_topology.cpp

```
240 TEST(stk_topology_how_to, check_for_positive_polarity)
241 {
242     stk::topology quad4Topology = stk::topology::QUAD_4;
243
244     ASSERT_EQ(8u, quad4Topology.num_permutations());
245     ASSERT_EQ(4u, quad4Topology.num_positive_permutations());
246
247     EXPECT_TRUE( quad4Topology.is_positive_polarity(0));
248     EXPECT_TRUE( quad4Topology.is_positive_polarity(1));
249     EXPECT_TRUE( quad4Topology.is_positive_polarity(2));
250     EXPECT_TRUE( quad4Topology.is_positive_polarity(3));
251     EXPECT_TRUE(!quad4Topology.is_positive_polarity(4));
252     EXPECT_TRUE(!quad4Topology.is_positive_polarity(5));
253     EXPECT_TRUE(!quad4Topology.is_positive_polarity(6));
254     EXPECT_TRUE(!quad4Topology.is_positive_polarity(7));
255 }
```

2.1.10 STK topology's `lexicographical_smallest_permutation` method

Listing 2.10 demonstrates the API for obtaining the smallest lexicographical permutation index. *The support for HEX_8, etc., only includes positive node-permutations.*

Listing 2.10: Example using `lexicographical_smallest_permutation`
../../../code/stk/stk_topology/tests_for_documentation/how_to_use_stk_topology.cpp

```
56  TEST(stk_topology_understanding, lexicographical_smallest_permutation)
57  {
58      {
59          unsigned triangle_node_ids[3] = {10, 8, 12};
60
61          stk::topology triangular_shell = stk::topology::SHELL_TRIANGLE_3;
62
63          unsigned gold_triangle_permutations[18]= {
64                  10, 8, 12,
65                  12, 10, 8,
66                  8, 12, 10, // lexicographical smallest permutation by node ids if considering
                        only positive permutations
67                  10, 12, 8,
68                  12, 8, 10,
69                  8, 10, 12  // lexicographical smallest permutation by node ids if considering
                        all permutations
70          };
71
72          verifyPermutationsForTriangle(triangular_shell, triangle_node_ids,
                gold_triangle_permutations);
```

```
73
74          bool usePositivePermutationsOnly = false;
75          unsigned permutation_index =
                triangular_shell.lexicographical_smallest_permutation(triangle_node_ids,
                usePositivePermutationsOnly);
76          unsigned gold_lexicographical_smallest_permutation_index = 5;
77          // driven by vertices, NOT mid-edge nodes
78          EXPECT_EQ(gold_lexicographical_smallest_permutation_index, permutation_index);
79
80          usePositivePermutationsOnly = true;
81          permutation_index =
                triangular_shell.lexicographical_smallest_permutation(triangle_node_ids,
                usePositivePermutationsOnly);
82          gold_lexicographical_smallest_permutation_index = 2;
83          // driven by vertices, NOT mid-edge nodes
84          EXPECT_EQ(gold_lexicographical_smallest_permutation_index, permutation_index);
85      }
86  }
```

2.1.11 STK topology's `lexicographical smallest permutation preserve polarity` method

Listing 2.11 demonstrates the API for obtaining the smallest lexicographical permutation index that matches the polarity of the input permutation

Listing 2.11: Example using `lexicographical_smallest_permutation_preserve_polarity` ../../../code/stk/stk_topology/tests_for_documentation/how_to_use_stk_topology.cpp

```
90  TEST(stk_topology_understanding, lexicographical_smallest_permutation_preserve_polarity)
91  {
92      {
93          stk::topology triangular_shell = stk::topology::SHELL_TRIANGLE_3;
94          unsigned shell_node_ids[3] = {10, 8, 12};
95          {
96              unsigned triangle_node_ids[3] = {12, 10, 8};
97
98              unsigned permutation_index =
                    triangular_shell.lexicographical_smallest_permutation_preserve_polarity(triangle_node_ids,
                    shell_node_ids);
99              unsigned expected_positive_permutation = 2;
100
101             EXPECT_EQ(expected_positive_permutation, permutation_index);
102             EXPECT_LT(expected_positive_permutation,
                    triangular_shell.num_positive_permutations());
103         }
104         {
105             unsigned triangle_node_ids[3] = {12, 8, 10};
106
107             unsigned permutation_index =
                    triangular_shell.lexicographical_smallest_permutation_preserve_polarity(triangle_node_ids,
                    shell_node_ids);
108             unsigned expected_negative_permutation = 5;
109
110             EXPECT_EQ(expected_negative_permutation, permutation_index);
111             EXPECT_GE(expected_negative_permutation,
                    triangular_shell.num_positive_permutations());
112         }
113     }
114 }
115
```

```
116 TEST(stk_topology_understanding, quad_lexicographical_smallest_permutation_preserve_polarity)
117 {
118     {
119         stk::topology quad_shell = stk::topology::SHELL_QUAD_4;
120         unsigned shell_node_ids[4] = {1, 2, 3, 4};
121         {
122             unsigned quad_node_ids[4] = {1, 2, 3, 4};
123 
124             unsigned permutation_index =
                 quad_shell.lexicographical_smallest_permutation_preserve_polarity(quad_node_ids,
                 shell_node_ids);
125             unsigned expected_positive_permutation = 0;
126 
127             EXPECT_EQ(expected_positive_permutation, permutation_index);
128             EXPECT_LT(expected_positive_permutation, quad_shell.num_positive_permutations());
129         }
130 
131         {
132             unsigned quad_node_ids[4] = {1, 4, 3, 2};
133 
134             unsigned permutation_index =
                 quad_shell.lexicographical_smallest_permutation_preserve_polarity(quad_node_ids,
                 shell_node_ids);
135             unsigned expected_negative_permutation = 4;
136 
137             EXPECT_EQ(expected_negative_permutation, permutation_index);
138             EXPECT_GE(expected_negative_permutation, quad_shell.num_positive_permutations());
139         }
140 
141         {
142             unsigned quad_node_ids[4] = {4, 2, 3, 1};
143 
144             unsigned permutation_index =
                 quad_shell.lexicographical_smallest_permutation_preserve_polarity(quad_node_ids,
                 shell_node_ids);
145             unsigned expected_invalid_permutation = 8;
146 
147             EXPECT_EQ(expected_invalid_permutation, permutation_index);
148             EXPECT_EQ(expected_invalid_permutation, quad_shell.num_permutations());
149         }
150     }
151 }
```

2.1.12 STK Topology's `sub_topology` methods

Listing 2.12 demonstrates the API for obtaining information about a topology's sub-topologies (sub-topologies define downward-connected entities; e.g., the face-rank sub-topology of HEX_20 is QUAD_8.).

Listing 2.12: Example using of `sub_topology`
../../../code/stk/stk_topology/tests_for_documentation/how_to_use_stk_topology.cpp

```
155 TEST(stk_topology_understanding, sub_topology)
156 {
157     stk::topology hex20 = stk::topology::HEX_20;
158     unsigned hex20Nodes[20] = {
159             0, 1, 2, 3,
160             4, 5, 6, 7,
161             8, 9, 10, 11,
162             12, 13, 14, 15,
163             16, 17, 18, 19
```

```
164     };
165
166     unsigned numFaces = hex20.num_sub_topology(stk::topology::FACE_RANK);
167     EXPECT_EQ(6u, numFaces);
168
169     unsigned faceIndex=2;
170     stk::topology top = hex20.sub_topology(stk::topology::FACE_RANK, faceIndex);
171     EXPECT_EQ(stk::topology::QUADRILATERAL_8, top);
172
173     unsigned nodeIdsFace[8];
174     hex20.sub_topology_nodes(hex20Nodes, stk::topology::FACE_RANK, faceIndex, nodeIdsFace);
175
176     unsigned goldIdsFace[8] = { 2, 3, 7, 6, 10, 15, 18, 14 };
177     for (unsigned i=0;i<hex20.face_topology(faceIndex).num_nodes();i++)
178     {
179         EXPECT_EQ(goldIdsFace[i], nodeIdsFace[i]);
180     }
181 }
```

2.1.13 STK Topology's `sides` methods

Listing 2.13 demonstrates the API for understanding sides in STK topologies. Note that for some topologies, *sides* differs in meaning from the Exodus [1] standard. For example, the number of sides on a shell-4 in Exodus is 6 (two faces, 4 edges) while the SHELL_QUAD_4 in stk_topology only counts the faces as sides, i.e., `num_sides()` returns 2.

Listing 2.13: Example for understanding sides in STK topology ../../../code/stk/stk_topology/tests_for_documentation/how_to_use_stk_topology.cpp

```
184 TEST(stk_topology_understanding, sides)
185 {
186     stk::topology hex20 = stk::topology::HEX_20;
187     EXPECT_EQ(6u, hex20.num_sides());
188
189     stk::topology quad8 = stk::topology::SHELL_QUADRILATERAL_8;
190     EXPECT_EQ(2u, quad8.num_sides());
191
192     stk::topology wedge = stk::topology::WEDGE_15;
193     EXPECT_EQ(5u, wedge.num_sides());
194     EXPECT_EQ(stk::topology::QUADRILATERAL_8, wedge.side_topology(0));
195     EXPECT_EQ(stk::topology::QUADRILATERAL_8, wedge.side_topology(1));
196     EXPECT_EQ(stk::topology::QUADRILATERAL_8, wedge.side_topology(2));
197     EXPECT_EQ(stk::topology::TRIANGLE_6, wedge.side_topology(3));
198     EXPECT_EQ(stk::topology::TRIANGLE_6, wedge.side_topology(4));
199
200 }
```

2.1.14 STK topology for a SuperElement

Listing 2.14 demonstrates the API for using super elements in STK Topology.

Listing 2.14: Example using a SuperElement with STK topology ../../../code/stk/stk_topology/tests_for_documentation/how_to_use_stk_topology.cpp

```
203 TEST(stk_topology_understanding, superelements)
204 {
205     unsigned eightNodes=8;
206     stk::topology validSuperElement = stk::create_superelement_topology(eightNodes);
207     EXPECT_TRUE(validSuperElement.is_superelement());
208     EXPECT_TRUE(stk::topology::ELEMENT_RANK == validSuperElement.rank());
209     EXPECT_EQ(eightNodes, validSuperElement.num_nodes());
210     EXPECT_EQ(0u, validSuperElement.num_edges());
211     EXPECT_EQ(0u, validSuperElement.num_faces());
212     EXPECT_EQ(0u, validSuperElement.num_permutations());
213     EXPECT_EQ(0u, validSuperElement.num_sides());
214     EXPECT_EQ(0u, validSuperElement.dimension());
215     EXPECT_EQ(stk::topology::INVALID_TOPOLOGY, validSuperElement.face_topology());
216     EXPECT_EQ(stk::topology::INVALID_TOPOLOGY, validSuperElement.edge_topology());
217     EXPECT_EQ(stk::topology::INVALID_TOPOLOGY, validSuperElement.base());
218     EXPECT_FALSE(validSuperElement.has_homogeneous_faces());
219     EXPECT_FALSE(validSuperElement.is_shell());
220
221     unsigned zeroNodes=0;
222     stk::topology invalidSuperElement = stk::create_superelement_topology(zeroNodes);
223     EXPECT_FALSE(invalidSuperElement.is_superelement());
224     EXPECT_TRUE(stk::topology::INVALID_RANK == invalidSuperElement.rank());
225     EXPECT_EQ(zeroNodes, invalidSuperElement.num_nodes());
226     EXPECT_EQ(0u, invalidSuperElement.num_edges());
227     EXPECT_EQ(0u, invalidSuperElement.num_faces());
228     EXPECT_EQ(0u, invalidSuperElement.num_permutations());
229     EXPECT_EQ(0u, invalidSuperElement.num_sides());
230     EXPECT_EQ(0u, invalidSuperElement.dimension());
231     EXPECT_EQ(stk::topology::INVALID_TOPOLOGY, invalidSuperElement.face_topology());
232     EXPECT_EQ(stk::topology::INVALID_TOPOLOGY, invalidSuperElement.edge_topology());
233     EXPECT_EQ(stk::topology::INVALID_TOPOLOGY, invalidSuperElement.base());
234     EXPECT_FALSE(invalidSuperElement.has_homogeneous_faces());
235     EXPECT_FALSE(invalidSuperElement.is_shell());
236 }
```

2.2 Mapping of Sierra topologies

Listing 2.15 compares four topology implementations found in Sierra: the Exodus Topology (defined by the name and number of nodes of the element), Ioss Topology, STK Topology, and the Cell (Shards) Topology. The test shows how a few elements compare for these implementations.

Listing 2.15: Example for understanding various Sierra topologies
../../../code/stk/stk_topology/tests_for_documentation/understanding_various_topologies.cpp

```
67
68 void setUpMappingsToTest(std::vector<TopologyMapper>& topologyMappings)
69 {
70     std::string exodusName;
71     int exodusNumNodes=-1;
72     std::string iossTopologyName;
73     stk::topology stkTopology;
74     stk::mesh::CellTopology shardsTopology;
75
76     exodusName="sphere";
77     exodusNumNodes=1;
78     iossTopologyName="sphere";
79     stkTopology=stk::topology::PARTICLE;
80     shardsTopology=stk::mesh::CellTopology(shards::getCellTopologyData< shards::Particle >());
81     topologyMappings.push_back(TopologyMapper(exodusName, exodusNumNodes, iossTopologyName,
            stkTopology, shardsTopology));
```

```
82
83      exodusName="BEam";
84      exodusNumNodes=3;
85      iossTopologyName="bar3";
86      stkTopology=stk::topology::BEAM_3;
87      shardsTopology=stk::mesh::CellTopology(shards::getCellTopologyData< shards::Beam<3> >());
88      topologyMappings.push_back(TopologyMapper(exodusName, exodusNumNodes, iossTopologyName,
                stkTopology, shardsTopology));
89
90      exodusName="Tri";
91      exodusNumNodes=3;
92      iossTopologyName="trishell3";
93      stkTopology=stk::topology::SHELL_TRIANGLE_3;
94      shardsTopology=stk::mesh::CellTopology(shards::getCellTopologyData<
                shards::ShellTriangle<3> >());
95      topologyMappings.push_back(TopologyMapper(exodusName, exodusNumNodes, iossTopologyName,
                stkTopology, shardsTopology));
96
97      exodusName="hex";
98      exodusNumNodes=20;
99      iossTopologyName="hex20";
100     stkTopology=stk::topology::HEXAHEDRON_20;
101     shardsTopology=stk::mesh::CellTopology(shards::getCellTopologyData<
                shards::Hexahedron<20> >());
102     topologyMappings.push_back(TopologyMapper(exodusName, exodusNumNodes, iossTopologyName,
                stkTopology, shardsTopology));
103 }
104
105 TEST(Understanding, sierra_topologies)
106 {
107     int spatialDim = 3;
108     std::vector<TopologyMapper> topologyMappings;
109     setUpMappingsToTest(topologyMappings);
110
111     size_t numMappings = topologyMappings.size();
112
113     createIossElementRegistryForKnownElementTopologies();
114
115     for (size_t i=0;i<numMappings;i++)
116     {
117         TopologyMapper goldValues = topologyMappings[i];
118
119         std::string fixedExodusName = Ioss::Utils::fixup_type(topologyMappings[i].exodusName,
                topologyMappings[i].exodusNumNodes, spatialDim);
120         Ioss::ElementTopology *iossTopology = Ioss::ElementTopology::factory(fixedExodusName,
                true);
121         ASSERT_TRUE(iossTopology != NULL);
122         EXPECT_EQ(goldValues.iossTopologyName, iossTopology->name());
123
124         stk::topology mappedStkTopologyFromIossTopology =
                stk::io::map_ioss_topology_to_stk(iossTopology);
125         EXPECT_EQ(goldValues.stkTopology, mappedStkTopologyFromIossTopology);
126
127         stk::mesh::CellTopology mappedShardsTopologyFromStkTopology =
                stk::mesh::get_cell_topology(mappedStkTopologyFromIossTopology);
128         EXPECT_EQ(goldValues.shardsTopology, mappedShardsTopologyFromStkTopology);
129
130         stk::topology mappedStkTopologyFromShards =
                stk::mesh::get_topology(mappedShardsTopologyFromStkTopology, spatialDim);
131         EXPECT_EQ(goldValues.stkTopology, mappedStkTopologyFromShards);
132     }
133 }
```

Some client applications still heavily use shards topologies with STK Mesh. To maintain support for this capability, STK Mesh provides a fast mapping between shards and stk_topology (see

listing 2.16).

Listing 2.16: Mapping of shards::CellTopologies to stk::topologies provided by stk::mesh::get_cell_topology() ../../../code/stk/stk_mesh/stk_mesh/base/MetaData.cpp

```
1039 CellTopology get_cell_topology(stk::topology t)
1040 {
1041   switch(t())
1042   {
1043   case stk::topology::NODE:
1044       return CellTopology( shards::getCellTopologyData< shards::Node                  >() );
1045   case stk::topology::LINE_2:
1046       return CellTopology( shards::getCellTopologyData< shards::Line<2>               >() );
1047   case stk::topology::LINE_3:
1048       return CellTopology( shards::getCellTopologyData< shards::Line<3>               >() );
1049   case stk::topology::TRI_3:
1050       return CellTopology( shards::getCellTopologyData< shards::Triangle<3>           >() );
1051   case stk::topology::TRI_4:
1052       return CellTopology( shards::getCellTopologyData< shards::Triangle<4>           >() );
1053   case stk::topology::TRI_6:
1054       return CellTopology( shards::getCellTopologyData< shards::Triangle<6>           >() );
1055   case stk::topology::QUAD_4:
1056       return CellTopology( shards::getCellTopologyData< shards::Quadrilateral<4>      >() );
1057   case stk::topology::QUAD_8:
1058       return CellTopology( shards::getCellTopologyData< shards::Quadrilateral<8>      >() );
1059   case stk::topology::QUAD_9:
1060       return CellTopology( shards::getCellTopologyData< shards::Quadrilateral<9>      >() );
1061   case stk::topology::PARTICLE:
1062       return CellTopology( shards::getCellTopologyData< shards::Particle              >() );
1063   case stk::topology::LINE_2_1D:
1064       return CellTopology( shards::getCellTopologyData< shards::Line<2>               >() );
1065   case stk::topology::LINE_3_1D:
1066       return CellTopology( shards::getCellTopologyData< shards::Line<3>               >() );
1067   case stk::topology::BEAM_2:
1068       return CellTopology( shards::getCellTopologyData< shards::Beam<2>               >() );
1069   case stk::topology::BEAM_3:
1070       return CellTopology( shards::getCellTopologyData< shards::Beam<3>               >() );
1071   case stk::topology::SHELL_LINE_2:
1072       return CellTopology( shards::getCellTopologyData< shards::ShellLine<2>          >() );
1073   case stk::topology::SHELL_LINE_3:
1074       return CellTopology( shards::getCellTopologyData< shards::ShellLine<3>          >() );
1075   case stk::topology::TRI_3_2D:
1076       return CellTopology( shards::getCellTopologyData< shards::Triangle<3>           >() );
1077   case stk::topology::TRI_4_2D:
1078       return CellTopology( shards::getCellTopologyData< shards::Triangle<4>           >() );
1079   case stk::topology::TRI_6_2D:
1080       return CellTopology( shards::getCellTopologyData< shards::Triangle<6>           >() );
1081   case stk::topology::QUAD_4_2D:
1082       return CellTopology( shards::getCellTopologyData< shards::Quadrilateral<4>      >() );
1083   case stk::topology::QUAD_8_2D:
1084       return CellTopology( shards::getCellTopologyData< shards::Quadrilateral<8>      >() );
1085   case stk::topology::QUAD_9_2D:
1086       return CellTopology( shards::getCellTopologyData< shards::Quadrilateral<9>      >() );
1087   case stk::topology::SHELL_TRI_3:
1088       return CellTopology( shards::getCellTopologyData< shards::ShellTriangle<3>      >() );
1089   case stk::topology::SHELL_TRI_4:break;
1090     //NOTE: shards does not define a topology for a 4-noded triangular shell
1091     //return CellTopology( shards::getCellTopologyData< shards::ShellTriangle<4>    >() );
1092   case stk::topology::SHELL_TRI_6:
1093       return CellTopology( shards::getCellTopologyData< shards::ShellTriangle<6>      >() );
1094   case stk::topology::SHELL_QUAD_4:
1095       return CellTopology( shards::getCellTopologyData< shards::ShellQuadrilateral<4> >() );
1096   case stk::topology::SHELL_QUAD_8:
1097       return CellTopology( shards::getCellTopologyData< shards::ShellQuadrilateral<8> >() );
1098   case stk::topology::SHELL_QUAD_9:
1099       return CellTopology( shards::getCellTopologyData< shards::ShellQuadrilateral<9> >() );
1100   case stk::topology::TET_4:
```

```
1101        return CellTopology( shards::getCellTopologyData< shards::Tetrahedron<4>        >() );
1102    case stk::topology::TET_8:
1103        return CellTopology( shards::getCellTopologyData< shards::Tetrahedron<8>        >() );
1104    case stk::topology::TET_10:
1105        return CellTopology( shards::getCellTopologyData< shards::Tetrahedron<10>       >() );
1106    case stk::topology::TET_11:
1107        return CellTopology( shards::getCellTopologyData< shards::Tetrahedron<11>       >() );
1108    case stk::topology::PYRAMID_5:
1109        return CellTopology( shards::getCellTopologyData< shards::Pyramid<5>            >() );
1110    case stk::topology::PYRAMID_13:
1111        return CellTopology( shards::getCellTopologyData< shards::Pyramid<13>           >() );
1112    case stk::topology::PYRAMID_14:
1113        return CellTopology( shards::getCellTopologyData< shards::Pyramid<14>           >() );
1114    case stk::topology::WEDGE_6:
1115        return CellTopology( shards::getCellTopologyData< shards::Wedge<6>              >() );
1116    case stk::topology::WEDGE_15:
1117        return CellTopology( shards::getCellTopologyData< shards::Wedge<15>             >() );
1118    case stk::topology::WEDGE_18:
1119        return CellTopology( shards::getCellTopologyData< shards::Wedge<18>             >() );
1120    case stk::topology::HEX_8:
1121        return CellTopology( shards::getCellTopologyData< shards::Hexahedron<8>         >() );
1122    case stk::topology::HEX_20:
1123        return CellTopology( shards::getCellTopologyData< shards::Hexahedron<20>        >() );
1124    case stk::topology::HEX_27:
1125        return CellTopology( shards::getCellTopologyData< shards::Hexahedron<27>        >() );
1126    default: break;
1127    }
1128    return CellTopology(NULL);
1129 }
```

This page intentionally left blank.

Chapter 3

STK Fields

A STK *field* is a data structure that defines values associated with entities, such as temperatures, coordinates, or stress. A field can be defined over the whole mesh or a subset of the mesh (typically defined by a list of parts). STK Mesh currently manages STK field creation, storage, retrieval and field data memory allocation. Fields are managed by entity rank (node, edge, face, element, etc.). Fields can have the same name as long as they are defined on different entity ranks.

The following code listings demonstrate some common usage of fields:

- Scalar, vector, and tensor fields
- Fields on nodes or on elements
- Fields allocated for the entire mesh
- Fields allocated for only part of the mesh
- Fields with constant size across the mesh
- Fields with variable size per part
- Multi-state fields
- Communicate field data

In each example, the general flow of execution is as follows:

1. Declare and initialize `stk::mesh::MetaData`: declare fields and parts
2. Declare and initialize `stk::mesh::BulkData`: create elements and nodes
3. Initialize, access and/or test field-data.

3.1 Example STK fields usage

Listing 3.1: Examples of constant-size whole-mesh field usage
../../../code/stk/stk_mesh/testsForDocumentation/useSimpleFields.cpp

```
69 TEST(stkMeshHowTo, useSimpleFields)
70 {
71     stk::mesh::MetaData metaData(SpatialDimension::three, stk::mesh::entity_rank_names());
72
73     typedef stk::mesh::Field<double> ScalarField;
74     typedef stk::mesh::Field<double, stk::mesh::Cartesian3d> VectorField;
```

```
75      ScalarField& pressureField =
                metaData.declare_field<ScalarField>(stk::topology::ELEM_RANK, "pressure");
76      VectorField& displacementsField =
                metaData.declare_field<VectorField>(stk::topology::NODE_RANK, "displacements");
77
78      double initialPressureValue = 4.4;
79      stk::mesh::put_field_on_entire_mesh_with_initial_value(pressureField,
                &initialPressureValue);
80      stk::mesh::put_field_on_entire_mesh(displacementsField);
81
82      stk::mesh::BulkData mesh(metaData, MPI_COMM_WORLD);
83      create_two_tet_element_mesh(mesh);
84
85      const stk::mesh::BucketVector& nodeBuckets = mesh.buckets(stk::topology::NODE_RANK);
86      EXPECT_TRUE(!nodeBuckets.empty());
87      for(size_t bucketIndex=0; bucketIndex<nodeBuckets.size(); bucketIndex++)
88      {
89          const stk::mesh::Bucket& bucket = *nodeBuckets[bucketIndex];
90          double* displacementDataForBucket = stk::mesh::field_data(displacementsField, bucket);
91          EXPECT_GT(bucket.size(), 0u);
92          for(size_t nodeIndex=0; nodeIndex<bucket.size(); nodeIndex++)
93          {
94              unsigned numValuesPerNode =
                    stk::mesh::field_scalars_per_entity(displacementsField, bucket);
95              EXPECT_EQ(SpatialDimension::three, numValuesPerNode);
96              for(unsigned i=0; i<numValuesPerNode; i++)
97              {
98                  EXPECT_EQ(0.0, displacementDataForBucket[nodeIndex*numValuesPerNode + i]);
99                  displacementDataForBucket[nodeIndex*numValuesPerNode + i] = 99.9;
100             }
101         }
102     }
103
104     stk::mesh::Entity elem1 = mesh.get_entity(stk::topology::ELEM_RANK, 1);
105     double* pressureFieldDataForElem1 = stk::mesh::field_data(pressureField, elem1);
106     EXPECT_EQ(initialPressureValue, *pressureFieldDataForElem1);
107
108     stk::mesh::Entity elem2 = mesh.get_entity(stk::topology::ELEM_RANK, 2);
109     double* pressureFieldDataForElem2 = stk::mesh::field_data(pressureField, elem2);
110     EXPECT_EQ(initialPressureValue, *pressureFieldDataForElem2);
111 }
```

Multidimensional fields (including 'vector' fields) must be declared by passing a second type parameter into the field's templated parameter list; failure to do so will result in the instantiation of a scalar field.

Listing 3.2: Example of incorrect vector field declaration ../../../code/stk/stk_mesh/testsForDocumentation/useSimpleFields.cpp

```
127 TEST(stkMeshHowTo, declareVectorFields_putFieldIgnoresLength)
128 {
129     stk::mesh::MetaData metaData(SpatialDimension::three, stk::mesh::entity_rank_names());
130
131     typedef stk::mesh::Field<double, stk::mesh::Cartesian3d> VectorField;
132     VectorField& velocities = metaData.declare_field<VectorField>(stk::topology::NODE_RANK,
                "velocities");
133
134     typedef stk::mesh::Field<double> BadVectorField;
135     BadVectorField& displacements =
                metaData.declare_field<BadVectorField>(stk::topology::NODE_RANK,
                "displacements");
136
137     unsigned fieldLength = 3;
138     stk::mesh::put_field(velocities, metaData.universal_part(), fieldLength);
139     stk::mesh::put_field(displacements, metaData.universal_part(), fieldLength);
```

```
140
141     stk::mesh::BulkData mesh(metaData, MPI_COMM_WORLD);
142     create_single_tet_element(mesh);
143
144     stk::mesh::Entity node1 = mesh.get_entity(stk::topology::NODE_RANK, 1);
145     EXPECT_NE(stk::mesh::field_scalars_per_entity(velocities, node1),
                stk::mesh::field_scalars_per_entity(displacements, node1));
146 }
```

Listing 3.3: Examples of how to get fields by name ../../../code/stk/stk_mesh/testsForDocumentation/howToGetFields.cpp

```
47 TEST(stkMeshHowTo, getFields)
48 {
49     stk::mesh::MetaData metaData(SpatialDimension::three);
50
51     typedef stk::mesh::Field<double> ScalarField;
52     typedef stk::mesh::Field<double, stk::mesh::Cartesian3d> VectorField;
53
54     const std::string pressureFieldName = "pressure";
55     const std::string displacementsFieldName = "displacements";
56     ScalarField *pressureField =
                &metaData.declare_field<ScalarField>(stk::topology::ELEM_RANK,
                pressureFieldName);
57     VectorField *displacementsField =
                &metaData.declare_field<VectorField>(stk::topology::NODE_RANK,
                displacementsFieldName);
58     metaData.commit();
59
60     EXPECT_EQ(pressureField, metaData.get_field<ScalarField>(stk::topology::ELEM_RANK,
                pressureFieldName));
61     EXPECT_EQ(pressureField, metaData.get_field(stk::topology::ELEM_RANK, pressureFieldName));
62
63     EXPECT_EQ(displacementsField, metaData.get_field<VectorField>(stk::topology::NODE_RANK,
                displacementsFieldName));
64     EXPECT_EQ(displacementsField, metaData.get_field(stk::topology::NODE_RANK,
                displacementsFieldName));
65 }
```

Listing 3.4: Examples of using fields that are variable-size and defined on only a subset of the mesh ../../../code/stk/stk_mesh/testsForDocumentation/useAdvancedFields.cpp

```
50 TEST(stkMeshHowTo, useAdvancedFields)
51 {
52     const unsigned spatialDimension = 3;
53     stk::mesh::MetaData metaData(spatialDimension, stk::mesh::entity_rank_names());
54
55     typedef stk::mesh::Field<double, stk::mesh::Cartesian> VectorField;
56     typedef stk::mesh::Field<double, stk::mesh::FullTensor36> TensorField;
57     TensorField& tensorField = metaData.declare_field<TensorField>(stk::topology::ELEM_RANK,
                "tensor");
58     VectorField& variableSizeField =
                metaData.declare_field<VectorField>(stk::topology::ELEM_RANK,
                "variableSizeField");
59
60     stk::mesh::Part &tetPart = metaData.declare_part_with_topology("tetElementPart",
                stk::topology::TET_4);
61     stk::mesh::Part &hexPart = metaData.declare_part_with_topology("hexElementPart",
                stk::topology::HEX_8);
62
63     double initialTensorValue[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
64     stk::mesh::put_field_on_entire_mesh_with_initial_value(tensorField, initialTensorValue);
65
66     double initialVectorValue[] = {1, 2, 3, 4, 5, 6, 7, 8};
```

```
67      const unsigned nodesPerTet = 4;
68      stk::mesh::put_field(variableSizeField, tetPart, nodesPerTet, initialVectorValue);
69      const unsigned nodesPerHex = 8;
70      stk::mesh::put_field(variableSizeField, hexPart, nodesPerHex, initialVectorValue);
71
72      metaData.commit();
73      stk::mesh::BulkData mesh(metaData, MPI_COMM_WORLD);
74      mesh.modification_begin();
75      stk::mesh::EntityId tetId = 1;
76      stk::mesh::EntityIdVector tetNodes {1, 2, 3, 4};
77      stk::mesh::Entity tetElem=stk::mesh::declare_element(mesh, tetPart, tetId, tetNodes);
78      stk::mesh::EntityId hexId = 2;
79      stk::mesh::EntityIdVector hexNodes {5, 6, 7, 8, 9, 10, 11, 12};
80      stk::mesh::Entity hexElem=stk::mesh::declare_element(mesh, hexPart, hexId, hexNodes);
81      mesh.modification_end();
82
83      const unsigned tensor_scalars_per_hex = stk::mesh::field_scalars_per_entity(tensorField,
                hexElem);
84      const unsigned tensor_scalars_per_tet = stk::mesh::field_scalars_per_entity(tensorField,
                tetElem);
85
86      EXPECT_EQ(tensor_scalars_per_hex, tensor_scalars_per_tet);
87      const unsigned tensor_enum_size = stk::mesh::FullTensor36::Size;
88      EXPECT_EQ(tensor_scalars_per_hex, tensor_enum_size);
89
90      double* tensorData = stk::mesh::field_data(tensorField, hexElem);
91      for(unsigned i=0; i<tensor_scalars_per_hex; i++)
92      {
93          EXPECT_EQ(initialTensorValue[i], tensorData[i]);
94      }
95
96      const unsigned scalars_per_tet = stk::mesh::field_scalars_per_entity(variableSizeField,
                tetElem);
97      EXPECT_EQ(nodesPerTet, scalars_per_tet);
98
99      const unsigned scalars_per_hex = stk::mesh::field_scalars_per_entity(variableSizeField,
                hexElem);
100     EXPECT_EQ(nodesPerHex, scalars_per_hex);
101
102     double* vectorHexData = stk::mesh::field_data(variableSizeField, hexElem);
103     for(unsigned i=0; i<scalars_per_hex; i++)
104     {
105         EXPECT_EQ(initialVectorValue[i], vectorHexData[i]);
106     }
107
108     double* vectorTetData = stk::mesh::field_data(variableSizeField, tetElem);
109     for(unsigned i=0; i<scalars_per_tet; i++)
110     {
111         EXPECT_EQ(initialVectorValue[i], vectorTetData[i]);
112     }
113 }
```

Some application time-stepping algorithms use multi-state fields to assist with separating and updating the field values for time-step n, $n-1$, $n+1$, etc. STK Mesh supports fields with up to 6 states.

Listing 3.5: Examples of multi-state field usage
../../../code/stk/stk_mesh/testsForDocumentation/useMultistateFields.cpp

```
49 TEST(stkMeshHowTo, useMultistateField)
50 {
51     const unsigned spatialDimension = 3;
52     stk::mesh::MetaData metaData(spatialDimension, stk::mesh::entity_rank_names());
53
54     typedef stk::mesh::Field<double> ScalarField;
```

```
55      const unsigned numStates = 2;
56      ScalarField& temperatureFieldStateNp1 =
                metaData.declare_field<ScalarField>(stk::topology::NODE_RANK, "temperature",
                numStates);
57
58      double initialTemperatureValue = 1.0;
59      stk::mesh::put_field_on_entire_mesh_with_initial_value(temperatureFieldStateNp1,
                &initialTemperatureValue);
60
61      metaData.commit();
62      stk::mesh::BulkData mesh(metaData, MPI_COMM_WORLD);
63      mesh.modification_begin();
64      stk::mesh::EntityId nodeId = 1;
65      stk::mesh::Entity node = mesh.declare_entity(stk::topology::NODE_RANK, nodeId);
66      mesh.modification_end();
67
68      EXPECT_EQ(stk::mesh::StateNP1, temperatureFieldStateNp1.state());
69      double* temperatureStateNp1 = stk::mesh::field_data(temperatureFieldStateNp1, node);
70      EXPECT_EQ(initialTemperatureValue, *temperatureStateNp1);
71      double newTemperatureValue = 2.0;
72      *temperatureStateNp1 = newTemperatureValue;
73
74      ScalarField& temperatureFieldStateN =
                temperatureFieldStateNp1.field_of_state(stk::mesh::StateN);
75      double* temperatureStateN = stk::mesh::field_data(temperatureFieldStateN, node);
76      EXPECT_EQ(initialTemperatureValue, *temperatureStateN);
77
78      mesh.update_field_data_states();
79
80      temperatureStateN = stk::mesh::field_data(temperatureFieldStateN, node);
81      EXPECT_EQ(newTemperatureValue, *temperatureStateN);
82  }
```

This page intentionally left blank.

Chapter 4

STK IO

4.1 STK IO: usage examples

STK IO is a module available for reading from and writing to Exodus [1] files (and other formats) into and out of STK Mesh. This section gives examples of how to use STK IO (referred hereon as STK Mesh IO Broker).

4.1.1 Reading mesh data to create a STK Mesh

The first example shows how to read mesh data from a file and create a STK Mesh corresponding to that mesh data. A STK Part will be created for each element block, nodeset, and sideset on the input mesh file and the name of the corresponding part will be the same as the name of the block or set in the mesh file.

Listing 4.1: Reading mesh data to create a STK mesh
../../../code/stk/stk_io/testsForDocumentation/readMesh.cpp

```
73      // ============================================================
74      //+ EXAMPLE:
75      //+ Read mesh data from the specified file.
76      stk::io::StkMeshIoBroker stkIo(communicator);
77      stkIo.add_mesh_database(mesh_name, stk::io::READ_MESH);
78
79      //+ Creates meta data; creates parts
80      stkIo.create_input_mesh();
81
82      //+ Any modifications to the meta data must be done here.
83      //+ This includes declaring fields.
84
85      //+ Commit the meta data and create the bulk data.
86      //+ populate the bulk data with data from the mesh file.
87      stkIo.populate_bulk_data();
88
89      // ============================================================
90      //+ VERIFICATION
91      //+ There should be:
92      //+ 4 parts corresponding to the 1 hex block and 3 shell blocks
93      stk::mesh::MetaData &meta = stkIo.meta_data();
94      stk::mesh::Part *invalid = NULL;
95      EXPECT_NE(invalid, meta.get_part("block_1"));
96      EXPECT_NE(invalid, meta.get_part("block_2"));
97      EXPECT_NE(invalid, meta.get_part("block_3"));
98      EXPECT_NE(invalid, meta.get_part("block_4"));
```

```
99
100       //+ 3 parts corresponding to the 3 nodesets.
101       EXPECT_NE(invalid, meta.get_part("nodelist_1"));
102       EXPECT_NE(invalid, meta.get_part("nodelist_2"));
103       EXPECT_NE(invalid, meta.get_part("nodelist_3"));
104
105       //+ 3 parts corresponding to the 3 sidesets.
106       EXPECT_NE(invalid, meta.get_part("surface_1"));
107       EXPECT_NE(invalid, meta.get_part("surface_2"));
108       EXPECT_NE(invalid, meta.get_part("surface_3"));
109
110
```

4.1.1.1 Face creation for input sidesets

Sidesets on volume elements where no shells are involved

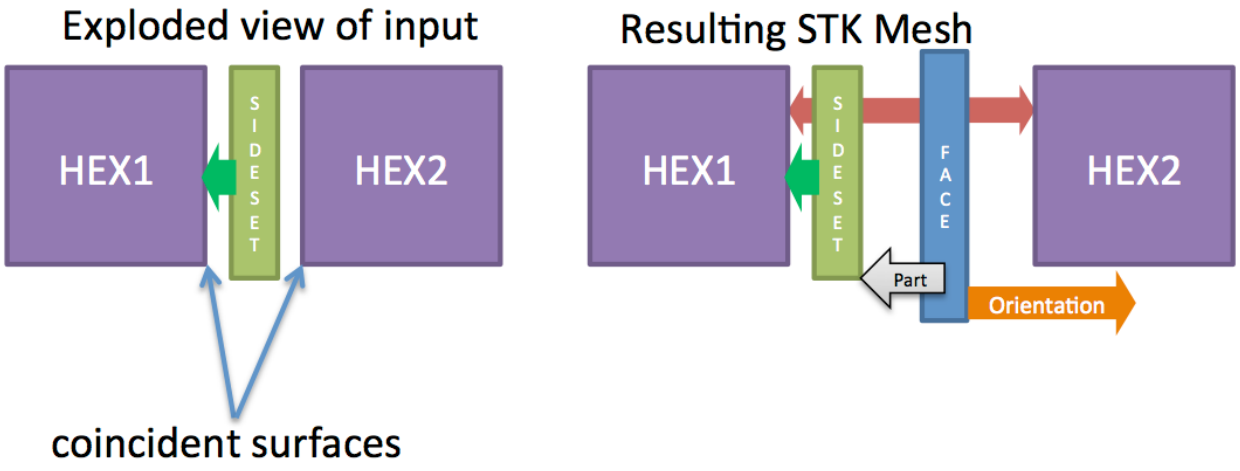

Figure 4.1: Sideset face creation in STK IO for 2 hexes.

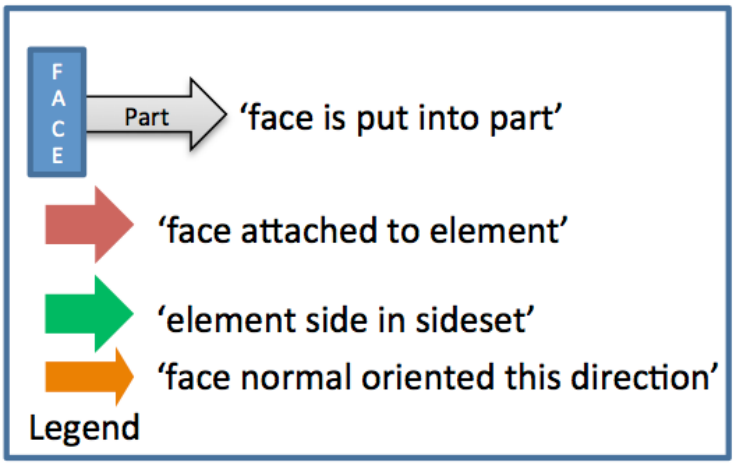

Figure 4.2: Legend for Sideset Face Creation

The simple case of reading in Exodus files with sidesets on an exposed or interior surfaces of volume elements (like hexes, tetrahedra, etc.) creates single faces on each surface during mesh read by StkMeshIOBroker. Additional sidesets on exposed or interior surfaces do not create additional faces but do add that face into additional STK parts.

When a face is created due to a sideset in Exodus, it is connected to all elements that share those nodes on a surface. So even if a sideset is present on an interior surface and has only one adjacent volume element, it will be connected to both volume elements that share that interior surface.

This includes doubly-sided sidesets with sides on the two adjacent interior surfaces on neighboring volume elements. In this case, only a single face that is connected to the two neighboring volume elements will be created but it will added to two STK parts. Whichever side of these coincident sidesets is listed first in the Exodus file will be created first, hence the orientation of that side will be used to set the orientation of the face. The SEACAS utility `ncdump` is useful in determining the ordering of sides and sidesets in Exodus files.

Figure 4.1 shows an example for 2 hexes with a sideset on the leftmost interior surface. Figure 4.2 shows the legend. Listing 4.2 documents the behavior and shows how to check.

Listing 4.2: Face creation during IO for one sideset between hexes
../../../code/stk/stk_doc_tests/stk_mesh/IOSidesetFaceCreation.cpp

```
55 bool is_positive_permutation(stk::mesh::BulkData & mesh,
56                                         stk::mesh::Entity face,
57                                         stk::mesh::Entity hex,
58                                         unsigned face_ordinal)
59 {
60     stk::topology faceTopology = mesh.bucket(face).topology();
61     stk::mesh::EntityVector face_nodes(mesh.num_nodes(face));
62     for (unsigned face_node_count=0; face_node_count < mesh.num_nodes(face);
               ++face_node_count) {
63         face_nodes[face_node_count] = mesh.begin_nodes(face)[face_node_count];
64     }
65     std::pair<bool, unsigned> permutation = stk::mesh::side_equivalent(mesh, hex,
               face_ordinal, face_nodes.data());
66
67     bool is_a_valid_permutation = permutation.first;
68     EXPECT_TRUE(is_a_valid_permutation);
69     bool is_positive_permutation = permutation.second <
               faceTopology.num_positive_permutations();
70     return is_positive_permutation;
71 }
72
73 TEST(StkMeshHowTo, StkIO2Hex1SidesetFaceCreation)
74 {
75     if (stk::parallel_machine_size(MPI_COMM_WORLD) == 1) {
76         //  -------  |S  -------              -------  |F  -------
77         //  |     |  |I  |     |              |     |  |A  |     |
78         //  |HEX1 5<-|D  4 HEX2| --STK-IO--> |HEX1 5<-|C->4 HEX2|
79         //  |     |  |E  |     |              |     |  |E  |     |
80         //  -------  |S  -------              -------  |   -------
81         //           |E                                |----> face is put into
82         //           |T                                        part surface_1
83         //                                             |---> orientation points outward
84         //                                                    from Hex1 face5
85
86         stk::io::StkMeshIoBroker stkMeshIoBroker(MPI_COMM_WORLD);
87         stkMeshIoBroker.add_mesh_database("ALA.e", stk::io::READ_MESH);
88         stkMeshIoBroker.create_input_mesh();
89         stkMeshIoBroker.populate_bulk_data();
```

```
90
91          stk::mesh::BulkData &mesh = stkMeshIoBroker.bulk_data();
92          stk::mesh::EntityVector all_faces;
93          stk::mesh::get_entities(mesh, stk::topology::FACE_RANK, all_faces);
94          std::sort(all_faces.begin(),all_faces.end());
95          unsigned expected_num_faces = 1;
96          ASSERT_EQ(expected_num_faces, all_faces.size());
97          size_t face_index = 0;
98          stk::mesh::Entity face = all_faces[face_index];
99          unsigned expected_connected_elements = 2;
100         ASSERT_EQ(expected_connected_elements, mesh.num_elements(face));
101
102         EXPECT_TRUE(mesh.bucket(face).member(*mesh.mesh_meta_data().get_part("surface_1")));
103
104         const stk::mesh::Entity * connected_elements = mesh.begin_elements(face);
105         const stk::mesh::ConnectivityOrdinal * which_side_of_element =
                mesh.begin_element_ordinals(face);
106
107         {
108             int element_count = 0;
109             stk::mesh::Entity hex_2 = connected_elements[element_count];
110             EXPECT_EQ(2u, mesh.identifier(hex_2));
111             unsigned expected_face_ordinal = 4;
112             EXPECT_EQ(expected_face_ordinal, which_side_of_element[element_count]);
113             EXPECT_FALSE(is_positive_permutation(
114                     mesh, face, hex_2, expected_face_ordinal));
115         }
116
117         {
118             int element_count = 1;
119             stk::mesh::Entity hex_1 = connected_elements[element_count];
120             EXPECT_EQ(1u, mesh.identifier(hex_1));
121             unsigned expected_face_ordinal = 5;
122             EXPECT_EQ(expected_face_ordinal, which_side_of_element[element_count]);
123             EXPECT_TRUE(is_positive_permutation(
124                     mesh, face, hex_1, expected_face_ordinal));
125         }
126
127     }
128 }
```

Sidesets on shell elements

Sides in sidesets can be created on either surface of a shell or both surfaces. If a single side is present in the Exodus file, a single face will be created and connected to the shell on a single surface. If two sides are present, two faces will be created with opposite permutations and individually connected to single distinct surfaces of the shell.

Figure 4.3 shows an example of two cases on a single shell. Figure 4.2 shows the legend.

Sidesets on stacked shell elements

On coincident shells, a maximum of two faces are ever created with opposite permutations, no matter how many sidesets are present. Extra sidesets cause parts to be added to the faces. If a single face is created, it is hooked to the same orientation of every coincident shell. If two faces are created, they are individually hooked to the same orientation of all coincident shells.

Sidesets on mixed volume and shell elements

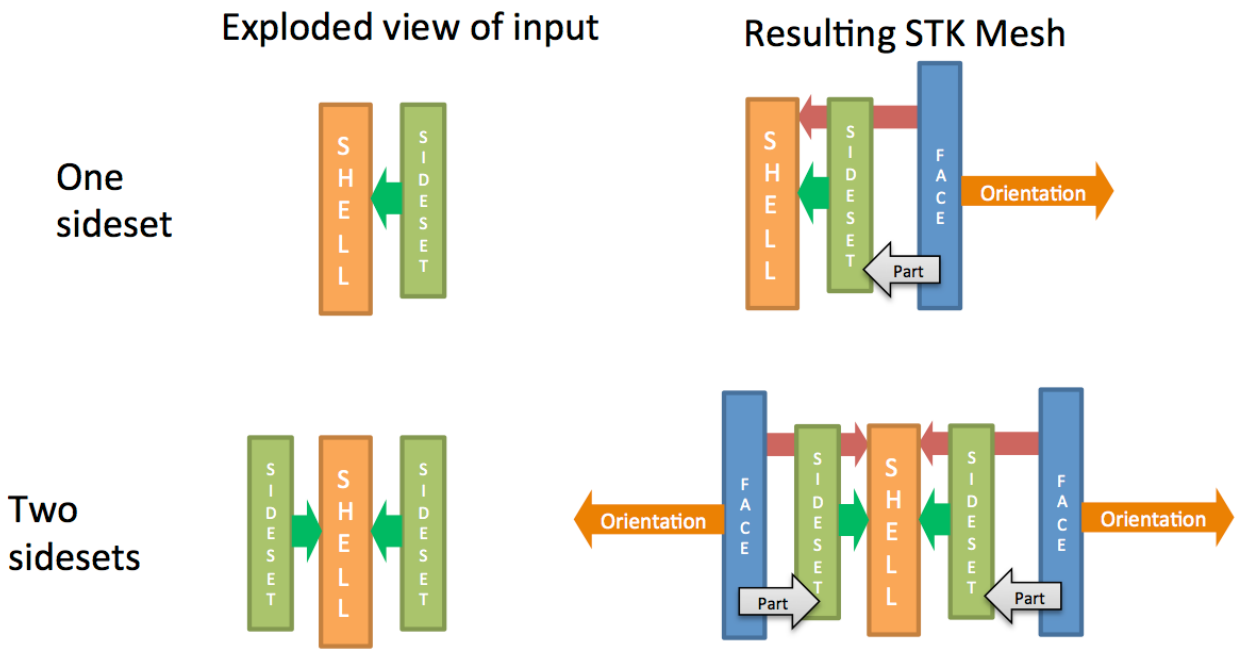

Figure 4.3: Sideset face creation in STK IO for one shell.

When shells are adjacent to volume elements, a maximum of two faces can be created (as opposed to single face with no shells present).

The first side in the first sideset (from the ordering in Exodus as checked by `ncdump`) determines the orientation of the face created for this surface on the element. If this side is on a volume element, it will be hooked to the opposite orientation of any and all coincident shells. If this side is on a shell element, it will be hooked to the same orientation of all other coincident shells but the opposite orientation of any adjacent surfaces on volume elements. If additional sides in sidesets are present in Exodus that would create faces that are already defined, additional parts will be created but not additional faces. If additional sides in sidesets would create a face on the opposing orientation of the shell, then it will be created and hooked to all other shell elements on that orientation and the opposite orientation of any adjacent surfaces on volume elements. Note that orientations of faces on volume elements are always outward directed.

Figure 4.4 an example of two shells between two hexes with three sidesets, only two faces are created. Figure 4.2 shows the legend. Listing 4.3 shows relevant code for checking the ordinals, permutations and parts.

Listing 4.3: Face creation during IO for shells between hexes with sidesets
../../../code/stk/stk_doc_tests/stk_mesh/IOSidesetFaceCreation.cpp

```
132 TEST(StkMeshHowTo, StkIO2Hex2Shell3SidesetFaceCreation)
133 {
134     if (stk::parallel_machine_size(MPI_COMM_WORLD) == 1) {
135         //  -------  |S |S| |S|  |S |S  -------
136         //  |     |  |I |H| |H|  |I |I  |     |
137         //  |HEX1 5<-|D |E| |E0<-|D |D->4 HEX2|
138         //  |     |  |E |L| |L|  |E |E  |     | |
139         //  -------  |S |L| |L|  |S |S  ------- |
```

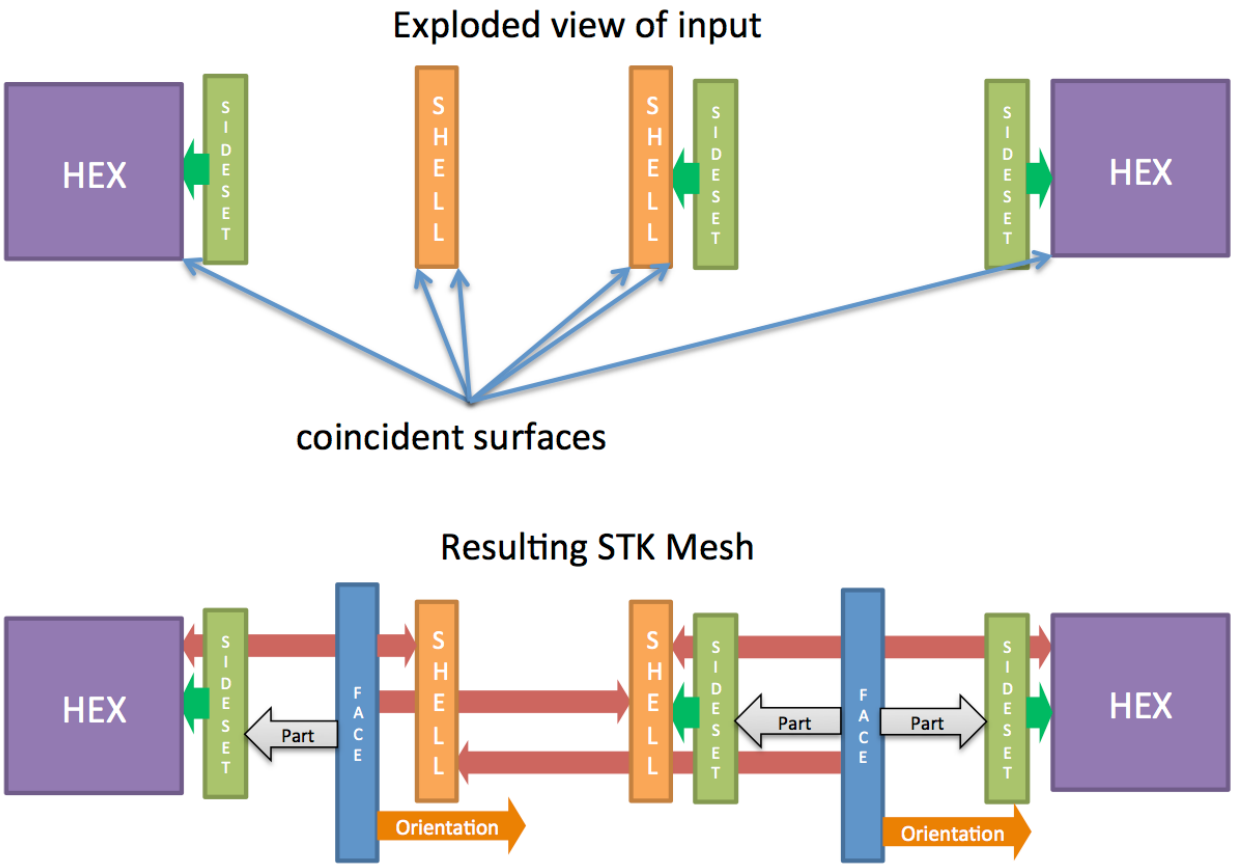

Figure 4.4: Sideset face creation in STK IO for a complicated example with stacked shells between two hex elements and multiple sidesets.

```
//           |E  3   4   |E |E          |
//           |T          |T |T         STK
//                                      IO
//                                       |
//                                       V
//
//  -------  |F  |S|  |S|             |F  -------
//  |     |  |A--|H|->1H|             |A  |     |
//  |HEX1 5<-|C->1E|  |E0<------------|C->4 HEX2|
//  |     |  |E  |L0<-|L|-------------|E  |     |
//  -------  |   |L|  |L|             |   -------
//           |    3    4             |
//           |---> orientation       |-->orientation
//           |---> in surface_1 part |-->in surface_2 and
//                                        surface_3 parts


stk::io::StkMeshIoBroker stkMeshIoBroker(MPI_COMM_WORLD);
stkMeshIoBroker.add_mesh_database("ALefLRA.e", stk::io::READ_MESH);
stkMeshIoBroker.create_input_mesh();
stkMeshIoBroker.populate_bulk_data();

stk::mesh::BulkData &mesh = stkMeshIoBroker.bulk_data();
stk::mesh::EntityVector all_faces;
stk::mesh::get_entities(mesh, stk::topology::FACE_RANK, all_faces);
std::sort(all_faces.begin(),all_faces.end());
unsigned expected_num_faces = 2;
```

```
167         ASSERT_EQ(expected_num_faces, all_faces.size());
168
169         size_t face_index = 0;
170         {
171             stk::mesh::Entity face = all_faces[face_index];
172             unsigned expected_connected_elements = 3;
173             ASSERT_EQ(expected_connected_elements, mesh.num_elements(face));
174
175
                EXPECT_TRUE(mesh.bucket(face).member(*mesh.mesh_meta_data().get_part("surface_1")));
176
177             const stk::mesh::Entity * connected_elements = mesh.begin_elements(face);
178             const stk::mesh::ConnectivityOrdinal * which_side_of_element =
                mesh.begin_element_ordinals(face);
179
180             {
181                 int element_count = 0;
182                 stk::mesh::Entity shell_3 = connected_elements[element_count];
183                 EXPECT_EQ(3u, mesh.identifier(shell_3));
184                 unsigned expected_face_ordinal = 1;
185                 EXPECT_EQ(expected_face_ordinal, which_side_of_element[element_count]);
186                 EXPECT_FALSE(is_positive_permutation(
187                         mesh, face, shell_3, expected_face_ordinal));
188             }
189             {
190                 int element_count = 1;
191                 stk::mesh::Entity shell_4 = connected_elements[element_count];
192                 EXPECT_EQ(4u, mesh.identifier(shell_4));
193                 unsigned expected_face_ordinal = 1;
194                 EXPECT_EQ(expected_face_ordinal, which_side_of_element[element_count]);
195                 EXPECT_FALSE(is_positive_permutation(
196                         mesh, face, shell_4, expected_face_ordinal));
197             }
198             {
199                 int element_count = 2;
200                 stk::mesh::Entity hex_1 = connected_elements[element_count];
201                 EXPECT_EQ(1u, mesh.identifier(hex_1));
202                 unsigned expected_face_ordinal = 5;
203                 EXPECT_EQ(expected_face_ordinal, which_side_of_element[element_count]);
204                 EXPECT_TRUE(is_positive_permutation(
205                         mesh, face, hex_1, expected_face_ordinal));
206             }
207         }
208
209         face_index = 1;
210         {
211             stk::mesh::Entity face = all_faces[face_index];
212             unsigned expected_connected_elements = 3;
213             ASSERT_EQ(expected_connected_elements, mesh.num_elements(face));
214
215
                EXPECT_TRUE(mesh.bucket(face).member(*mesh.mesh_meta_data().get_part("surface_2")));
216
                EXPECT_TRUE(mesh.bucket(face).member(*mesh.mesh_meta_data().get_part("surface_3")));
217
218             const stk::mesh::Entity * connected_elements = mesh.begin_elements(face);
219             const stk::mesh::ConnectivityOrdinal * which_side_of_element =
                mesh.begin_element_ordinals(face);
220
221             {
222                 int element_count = 0;
223                 stk::mesh::Entity shell_3 = connected_elements[element_count];
224                 EXPECT_EQ(3u, mesh.identifier(shell_3));
225                 unsigned expected_face_ordinal = 0;
226                 EXPECT_EQ(expected_face_ordinal, which_side_of_element[element_count]);
227                 EXPECT_FALSE(is_positive_permutation(
228                         mesh, face, shell_3, expected_face_ordinal));
229             }
```

```
230             {
231                 int element_count = 1;
232                 stk::mesh::Entity shell_4 = connected_elements[element_count];
233                 EXPECT_EQ(4u, mesh.identifier(shell_4));
234                 unsigned expected_face_ordinal = 0;
235                 EXPECT_EQ(expected_face_ordinal, which_side_of_element[element_count]);
236                 EXPECT_FALSE(is_positive_permutation(
237                         mesh, face, shell_4, expected_face_ordinal));
238             }
239             {
240                 int element_count = 2;
241                 stk::mesh::Entity hex_2 = connected_elements[element_count];
242                 EXPECT_EQ(2u, mesh.identifier(hex_2));
243                 unsigned expected_face_ordinal = 4;
244                 EXPECT_EQ(expected_face_ordinal, which_side_of_element[element_count]);
245                 EXPECT_TRUE(is_positive_permutation(mesh, face, hex_2,
            expected_face_ordinal));
246             }
247         }
248     }
249 }
```

STK IO Classic for Transition

To aid transition, we are documenting and preserving the old STK IO behavior for now. The old behavior is that every sideset creates a unique face. These faces are not hooked to other elements.

4.1.2 Reading mesh data to create a STK Mesh allowing StkMeshIoBroker to go out of scope

This example shows how to read mesh data from a file and create a STK Mesh corresponding to that mesh data while also allowing the StkMeshIoBroker to go out of scope without deleting the STK Mesh.

Listing 4.4: Reading mesh data to create a STK mesh using set bulk data../../../code/stk/stk_mesh/testsForDocumentation/createStkMeshAlt1.cpp

```
53 TEST(StkMeshHowTo, CreateStkMesh)
54 {
55   MPI_Comm communicator = MPI_COMM_WORLD;
56   if (stk::parallel_machine_size(communicator) != 1) { return; }
57   const std::string exodusFileName = "example.exo";
58
59   create_example_exodus_file(communicator, exodusFileName);
60   // Creation of STK Mesh objects.
61   // MetaData creates the universal_part, locally-owned part, and globally shared part.
62   const int spatialDim = 3;
63   stk::mesh::MetaData stkMeshMetaData(spatialDim);
64   stk::mesh::BulkData stkMeshBulkData(stkMeshMetaData, communicator);
65
66   // STK IO module will be described in separate chapter.
67   // It is used here to read the mesh data from the Exodus file and populate an STK Mesh.
68   // The order of the following lines in {} are important
69   {
70     stk::io::StkMeshIoBroker exodusFileReader(communicator);
71
72     // Inform STK IO which STK Mesh objects to populate later
73     exodusFileReader.set_bulk_data(stkMeshBulkData);
```

```
74
75      exodusFileReader.add_mesh_database(exodusFileName, stk::io::READ_MESH);
76
77      // Populate the MetaData which has the descriptions of the Parts and Fields.
78      exodusFileReader.create_input_mesh();
79
80      // Populate entities in STK Mesh from Exodus file
81      exodusFileReader.populate_bulk_data();
82    }
83
84    // Test if the STK Mesh has 512 elements. Other examples will discuss details below.
85    stk::mesh::Selector allEntities = stkMeshMetaData.universal_part();
86    std::vector<unsigned> entityCounts;
87    stk::mesh::count_entities(allEntities, stkMeshBulkData, entityCounts);
88    EXPECT_EQ(512u, entityCounts[stk::topology::ELEMENT_RANK]);
89    unlink(exodusFileName.c_str());
90  }
```

4.1.3 Reading mesh data to create a STK Mesh, delaying field allocations

This example is almost the same as the previous except it delays the allocation of field data so that the application can modify the mesh. If the field data is allocated prior to the mesh modification, the reordering and moving of field data memory may be expensive; if the field data allocation is delayed, no reordering or moving of memory is needed.

The field data memory allocation delay is accomplished by calling `populate_mesh()` and `populate_field_data()` instead of `populate_bulk_data()`. Any mesh modifications, for example, creating mesh edges or mesh faces is performed prior to calling `populate_field_data()`.

Listing 4.5: Reading mesh data to create a STK mesh; delay field allocation
../../../code/stk/stk_io/testsForDocumentation/readMeshDelayFieldAllocation.cpp

```
68      // ============================================================
69      //+ EXAMPLE:
70      //+ Read mesh data from the specified file.
71      stk::io::StkMeshIoBroker stkIo(communicator);
72      stkIo.add_mesh_database(mesh_name, stk::io::READ_MESH);
73
74      //+ Creates meta data; creates parts
75      stkIo.create_input_mesh();
76
77      //+ Any modifications to the meta data must be done here.
78      //+ This includes declaring fields.
79
80      //+ Commit the meta data and create the bulk data.
81      //+ populate the bulk data with data from the mesh file.
82      stkIo.populate_mesh();
83
84      //+ Application would call mesh modification here.
85      //+ for example, create_edges() or create_faces().
86
87      //+ Mesh modifications complete, allocate field data.
88      stkIo.populate_field_data();
89
90
```

4.1.4 Outputting STK Mesh

Listing 4.6: Writing a STK Mesh ../../../code/stk/stk_io/testsForDocumentation/howToWriteMesh.cpp

```
#include <unistd.h>
#include <gtest/gtest.h>
#include <stk_mesh/base/MetaData.hpp>
#include <stk_mesh/base/BulkData.hpp>
#include <stk_mesh/base/Comm.hpp>
#include <stk_io/StkMeshIoBroker.hpp>
#include <stk_unit_test_utils/ioUtils.hpp>
namespace
{

TEST(StkIoHowTo, WriteMesh)
{
    std::string filename = "output.exo";
    {
        stk::mesh::MetaData meta;
        stk::mesh::BulkData bulk(meta, MPI_COMM_WORLD);
        stk::unit_test_util::fill_mesh_using_stk_io("generated:1x1x4", bulk);

        stk::io::StkMeshIoBroker stkIo;
        stkIo.set_bulk_data(bulk);
        size_t outputFileIndex = stkIo.create_output_mesh(filename, stk::io::WRITE_RESULTS);
        stkIo.write_output_mesh(outputFileIndex);
        stkIo.write_defined_output_fields(outputFileIndex);
    }

    {
        stk::mesh::MetaData meta;
        stk::mesh::BulkData bulk(meta, MPI_COMM_WORLD);
        stk::unit_test_util::fill_mesh_using_stk_io(filename, bulk);

        std::vector<size_t> entityCounts;
        stk::mesh::comm_mesh_counts(bulk, entityCounts);
        EXPECT_EQ(4u, entityCounts[stk::topology::ELEM_RANK]);
    }

    unlink(filename.c_str());
}

}
```

4.1.5 Outputting results data from a STK Mesh

This example shows how an application can output the application's calculated field data to a results database.

Listing 4.7: Writing calculated field data to a results database ../../../code/stk/stk_io/testsForDocumentation/writeResults.cpp

```
      // ============================================================
      //+ EXAMPLE:
      //+ Read mesh data from the specified file.
      stk::io::StkMeshIoBroker stkIo(communicator);
      stkIo.add_mesh_database(mesh_name, stk::io::READ_MESH);

      //+ Creates meta data; creates parts
      stkIo.create_input_mesh();

      //+ Declare a field
```

```
92      //+ NOTE: Fields must be declared before "populate_bulk_data()" is called
93      //+       since it commits the meta data.
94      const std::string fieldName = "disp";
95      stk::mesh::Field<double> &field =
              stkIo.meta_data().declare_field<stk::mesh::Field<double>
              >(stk::topology::NODE_RANK, fieldName, 1);
96      stk::mesh::put_field(field, stkIo.meta_data().universal_part());
97
98      //+ commit the meta data and create the bulk data.
99      //+ populate the bulk data with data from the mesh file.
100     stkIo.populate_bulk_data();
101
102     // ============================================================
103     //+ Create results file. By default, all parts created from the input
104     //+ mesh will be written to the results output file.
105     size_t fh = stkIo.create_output_mesh(results_name, stk::io::WRITE_RESULTS);
106
107     //+ The field will be output to the results file with the default field name.
108     stkIo.add_field(fh, field);
109
110     std::vector<stk::mesh::Entity> nodes;
111     stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK, nodes);
112
113     // Iterate the application's execute loop five times and output
114     // field data each iteration.
115     for (int step=0; step < 5; step++) {
116       double time = step;
117
118       // Application execution...
119       double value = 10.0 * time;
120       for(size_t i=0; i<nodes.size(); i++) {
121         double *node_data = stk::mesh::field_data(field, nodes[i]);
122         *node_data = value;
123       }
124
125       //+ Output the field data calculated by the application.
126       stkIo.begin_output_step(fh, time);
127       stkIo.write_defined_output_fields(fh);
128       stkIo.end_output_step(fh);
129     }
130
```

4.1.6 Outputting a field with an alternative name to a results file

The client can specify a field name for results output that is different than the internally used STK Mesh field name. The results output field name is specified as the second argument to the `add_field()` function. The code excerpt shown below replaces line 108 in the previous example (Listing 4.7) to cause the name of the field on the output

Listing 4.8: Outputting a field with an alternative name
../../../code/stk/stk_io/testsForDocumentation/requestedResultsFieldName.cpp

```
105     //+ The field 'fieldName' will be output to the results file with the name
              'alternateFieldName'
106     std::string alternateFieldName("displacement");
107     stkIo.add_field(fh, field, alternateFieldName);
108
```

4.1.7 Outputting both results and restart data from a STK Mesh

The STK Mesh IO Broker class can output both results data and restart data. Currently, the only difference between results data and restart data is that a restart output will automatically output the multiple states of a multi-state field. If, for example, the application defines a three-state field named “disp”, then outputting this field to a restart database will result in the two newest states being output. On the restart database the variables will appear as “disp” and “disp.N.” Outputting this field to a results database will only output the data on the newest state as the variable “disp”. When the restart database is read back in, the variables will be restored back to the same states that were written.

The example below shows how an application can output both a results and restart database. The example shows both databases being written on each step, but this is not required – each file can specify its own output frequency.

Listing 4.9: Write results and restart
../../../code/stk/stk_io/testsForDocumentation/writeResultsAndRestart.cpp

```
84      // ============================================================
85      //+ EXAMPLE:
86      //+ Read mesh data from the specified file.
87      stk::io::StkMeshIoBroker stkIo(communicator);
88      stkIo.add_mesh_database(mesh_name, stk::io::READ_MESH);
89
90      //+ Creates meta data; creates parts
91      stkIo.create_input_mesh();
92
93      //+ Declare a three-state field
94      //+ NOTE: Fields must be declared before "populate_bulk_data()" is called
95      //+       since it commits the meta data.
96      const std::string fieldName = "disp";
97      stk::mesh::Field<double> &field =
              stkIo.meta_data().declare_field<stk::mesh::Field<double>
              >(stk::topology::NODE_RANK, fieldName, 3);
98      stk::mesh::put_field(field, stkIo.meta_data().universal_part());
99
100     const stk::mesh::Part& block_1 = *stkIo.meta_data().get_part("block_1");
101     //+ create a two-state field
102     stk::mesh::Field<double> &fooSubset = stkIo.meta_data().
103           declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK, "fooSubset",
              2);
104     stk::mesh::put_field(fooSubset, block_1);
105
106     //+ commit the meta data and create the bulk data.
107     //+ populate the bulk data with data from the mesh file.
108     stkIo.populate_bulk_data();
109
110     // ============================================================
111     //+ Create results file. By default, all parts created from the input
112     //+ mesh will be written to the results output file.
113     size_t results_fh = stkIo.create_output_mesh(results_name, stk::io::WRITE_RESULTS);
114
115     //+ Create restart file. By default, all parts created from the input
116     //+ mesh will be written to the results output file.
117     size_t restart_fh = stkIo.create_output_mesh(restart_name, stk::io::WRITE_RESTART);
118
119     //+ The field will be output to the results file with the default field name.
120     //+ Only the newest state will be output.
121     stkIo.add_field(results_fh, field);
122
```

```
123       //+ Output the field to the restart database also.
124       //+ The two newest states will be output.
125       stkIo.add_field(restart_fh, field);
126       stkIo.add_field(restart_fh, fooSubset);
127
128       std::vector<stk::mesh::Entity> nodes;
129       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK, nodes);
130
131       stk::mesh::FieldBase *statedFieldNp1 = field.field_state(stk::mesh::StateNP1);
132       stk::mesh::FieldBase *statedFieldN   = field.field_state(stk::mesh::StateN);
133       stk::mesh::FieldBase *statedFieldNm1 = field.field_state(stk::mesh::StateNM1);
134
135       // Iterate the application's execute loop five times and output
136       // field data each iteration.
137       for (int step=0; step < 5; step++) {
138         double time = step;
139
140         // Application execution...
141         double value = 10.0 * time;
142         for(size_t i=0; i<nodes.size(); i++) {
143           double *np1_data = static_cast<double*>(stk::mesh::field_data(*statedFieldNp1,
                nodes[i]));
144           *np1_data = value;
145           double *n_data   = static_cast<double*>(stk::mesh::field_data(*statedFieldN,
                nodes[i]));
146           *n_data   = value + 0.1;
147           double *nm1_data = static_cast<double*>(stk::mesh::field_data(*statedFieldNm1,
                nodes[i]));
148           *nm1_data = value + 0.2;
149         }
150
151         //+ Results output...
152         stkIo.begin_output_step(results_fh, time);
153         stkIo.write_defined_output_fields(results_fh);
154         stkIo.end_output_step(results_fh);
155
156         //+ Restart output...
157         stkIo.begin_output_step(restart_fh, time);
158         stkIo.write_defined_output_fields(restart_fh);
159         stkIo.end_output_step(restart_fh);
160       }
161
```

4.1.8 Writing multi-state fields to results output file

The previous example showed that a results file will only output the newest state of a multi-state field. However, it is possible to tell a results file to output multiple states from a multi-state field. Each state of the field must be registered individually. Since each state will have the same field name, the `add_field()` call must also specify the name to be used for the variable on the results database in order to get unique names for each state. The example below shows how to output all three states of a multi-state field to a results database.

Listing 4.10: Writing multi-state field to results output
../../../code/stk/stk_io/testsForDocumentation/usingResults.cpp

```
70      const std::string fieldName = "disp";
71      const std::string np1Name = fieldName+"NP1";
72      const std::string nName   = fieldName+"N";
73      const std::string nm1Name = fieldName+"Nm1";
```

```
74      {
75        // ============================================================
76        //+ INITIALIZATION
77        const std::string exodusFileName = "generated:1x1x8";
78        stk::io::StkMeshIoBroker stkIo(communicator);
79        size_t index = stkIo.add_mesh_database(exodusFileName, stk::io::READ_MESH);
80        stkIo.set_active_mesh(index);
81        stkIo.create_input_mesh();
82
83        //+ Declare a three-state field
84        //+ NOTE: Fields must be declared before "populate_bulk_data()" is called
85        //+       since it commits the meta data.
86        stk::mesh::Field<double> &field =
87          stkIo.meta_data().declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK,
88                                                                     fieldName, 3);
89        stk::mesh::put_field(field, stkIo.meta_data().universal_part());
90
91        stkIo.populate_bulk_data();
92
93        size_t fh =
94          stkIo.create_output_mesh(resultsFilename, stk::io::WRITE_RESULTS);
95
96        // ============================================================
97        //+ EXAMPLE
98        //+ Output each state of the multi-state field individually to results file
99        stk::mesh::FieldBase *statedFieldNp1 = field.field_state(stk::mesh::StateNP1);
100       stk::mesh::FieldBase *statedFieldN   = field.field_state(stk::mesh::StateN);
101       stk::mesh::FieldBase *statedFieldNm1 = field.field_state(stk::mesh::StateNM1);
102
103       std::vector<stk::mesh::Entity> nodes;
104       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK, nodes);
105
106       stkIo.add_field(fh, *statedFieldNp1, np1Name);
107       stkIo.add_field(fh, *statedFieldN,   nName);
108       stkIo.add_field(fh, *statedFieldNm1, nm1Name);
109
110       // Iterate the application's execute loop five times and output
111       // field data each iteration.
112       for (int step=0; step < 5; step++) {
113         double time = step;
114
115         // Application execution...
116         // Generate field data... (details omitted)
117
131        //+ Results output...
132        stkIo.begin_output_step(fh, time);
133        stkIo.write_defined_output_fields(fh);
134        stkIo.end_output_step(fh);
135       }
136
```

4.1.9 Writing multiple output files

The following example shows how to write multiple output files. Although different fields and global variables are written to each file in the example, the same field or global variable can be written to multiple files.

Listing 4.11: Writing multiple output files
../../../code/stk/stk_io/testsForDocumentation/writingMultipleOutputFiles.cpp

```
58      // ============================================================
```

```
59      //+ EXAMPLE -- Two results output files
60      stk::mesh::FieldBase *displacementField =
61        meta_data.get_field(stk::topology::NODE_RANK, displacementFieldName);
62
63      //+ For file one, set up results and global variables
64      size_t file1Handle = stkIo.create_output_mesh(resultsFilename1,
65                                                    stk::io::WRITE_RESULTS);
66      stkIo.add_field(file1Handle, *displacementField);
67      std::string globalVarNameFile1 = "eigenValue";
68      stkIo.add_global(file1Handle, globalVarNameFile1, Ioss::Field::REAL);
69
70      //+ For file two, set up results and global variables
71      size_t file2Handle = stkIo.create_output_mesh(resultsFilename2,
72                                                    stk::io::WRITE_RESULTS);
73      std::string nameOnOutputFile("deformations");
74      stkIo.add_field(file2Handle, *displacementField, nameOnOutputFile);
75      stk::mesh::FieldBase *velocityField = meta_data.get_field(stk::topology::NODE_RANK,
                velocityFieldName);
76      stkIo.add_field(file2Handle, *velocityField);
77      std::string globalVarNameFile2 = "kineticEnergy";
78      stkIo.add_global(file2Handle, globalVarNameFile2, Ioss::Field::REAL);
79
80      //+ Write output
81      double time = 0.0;
82      stkIo.begin_output_step(file1Handle, time);
83      stkIo.write_defined_output_fields(file1Handle);
84      const double globalVarValue1 = 13.0;
85      stkIo.write_global(file1Handle, globalVarNameFile1, globalVarValue1);
86      stkIo.end_output_step(file1Handle);
87
88      stkIo.begin_output_step(file2Handle, time);
89      stkIo.write_defined_output_fields(file2Handle);
90      const double globalVarValue2 = 14.0;
91      stkIo.write_global(file2Handle, globalVarNameFile2, globalVarValue2);
92      stkIo.end_output_step(file2Handle);
93
94
```

4.1.10 Outputting nodal variables on a subset of the nodes

By default, a nodal variable is assumed to be defined on all nodes of the mesh. If the variable does not exist on all nodes, then a value of zero will be output for those nodes. If a nodal variable is only defined on a few of the nodes of the mesh, this can increase the size of the mesh file since it is storing much more data than is required. There is an option in STK Mesh IO Broker to handle this case by creating one or more "nodesets" which consist of the nodes of the part or parts where the nodal variable is defined. The name of the nodeset will be the part name suffixed by "_n". For example, if the part is named "firset", the nodeset corresponding to the nodes of this part will be named "fireset_n".

Listing 4.12: Using a nodeset variable to output nodal fields defined on only a subset of the mesh ../../../code/stk/stk_io/testsForDocumentation/useNodesetDbVarForNodalField.cpp

```
75      // ============================================================
76      // INITIALIZATION
77      std::string s_elems_per_edge = Ioss::Utils::to_string(num_elems_per_edge);
78
79      //+ Create a generated mesh containg hexes and shells.
80      std::string input_filename = s_elems_per_edge + "x" +
```

```
81          s_elems_per_edge + "x" +
82          s_elems_per_edge + "|shell:xyzXYZ";
83
84        stk::io::StkMeshIoBroker stkIo(communicator);
85        stkIo.add_mesh_database(input_filename, "generated",
86                                stk::io::READ_MESH);
87        stkIo.create_input_mesh();
88
89        stk::mesh::MetaData &meta_data = stkIo.meta_data();
90        stk::mesh::Field<double> &temperature = meta_data.
91          declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK,
92                                                   appFieldName, 1);
93
94        // ============================================================
95        //+ Put the temperature field on the nodes of the shell parts.
96        const stk::mesh::PartVector &all_parts = meta_data.get_mesh_parts();
97        stk::mesh::Selector shell_subset;
98        for (size_t i=0; i < all_parts.size(); i++) {
99          const stk::mesh::Part *part = all_parts[i];
100         stk::topology topo = part->topology();
101         if (topo == stk::topology::SHELL_QUAD_4) {
102           stk::mesh::put_field(temperature, *part);
103         }
104       }
105
106       stkIo.populate_bulk_data();
107
108       // Create the output...
109       size_t fh = stkIo.create_output_mesh(resultsFilename, stk::io::WRITE_RESULTS);
110
111       //+ The "temperature" field will be output on nodesets consisting
112       //+ of the nodes of each part the field is defined on.
113       stkIo.use_nodeset_for_part_nodes_fields(fh, true);
114       stkIo.add_field(fh, temperature, dbFieldName);
115
116       std::vector<stk::mesh::Entity> nodes;
117       stk::mesh::get_entities(stkIo.bulk_data(),
118                               stk::topology::NODE_RANK, nodes);
119
120       // Add three steps to the database
121       // For each step, the value of the field is the value 'time'
122       for (size_t i=0; i < 3; i++) {
123         double time = i;
124
125         for(size_t inode=0; inode<nodes.size(); inode++) {
126           double *fieldDataForNode = stk::mesh::field_data(temperature, nodes[inode]);
127           if (fieldDataForNode)
128             *fieldDataForNode = time;
129         }
130
131         stkIo.begin_output_step(fh, time);
132         stkIo.write_defined_output_fields(fh);
133         stkIo.end_output_step(fh);
134       }
135       // Verification omitted...
136
```

4.1.11 Get number of time steps from a database

Listing 4.13: get num time steps
../../../code/stk/stk_io/testsForDocumentation/howToGetNumTimeSteps.cpp

```
26 TEST_F(ExodusFileWithVariables, queryingFileWithSingleTimeStep_NumTimeStepsEqualsOne)
```

```
{
    create_mesh_with_single_time_step(filename, get_comm());
    read_mesh(filename);
    EXPECT_EQ(1, stkIo.get_num_time_steps());
}

TEST_F(ExodusFileWithVariables, queryingFileWithoutTimeSteps_NumTimeStepsEqualsZero)
{
    stk::unit_test_util::create_mesh_without_time_steps(filename, get_comm());
    read_mesh(filename);
    EXPECT_EQ(0, stkIo.get_num_time_steps());
}

TEST_F(ExodusFileWithVariables, readDefinedInputFieldsFromInvalidTimeStep_throws)
{
    create_mesh_with_single_time_step(filename, get_comm());
    read_mesh(filename);
    EXPECT_THROW(stkIo.read_defined_input_fields(3), std::exception);
}

TEST_F(ExodusFileWithVariables, readDefinedInputFields_throws)
{
    stk::unit_test_util::create_mesh_without_time_steps(filename, get_comm());
    read_mesh(filename);
    EXPECT_THROW(stkIo.read_defined_input_fields(1), std::exception);
}
```

4.1.12 Reading sequenced fields from a database

Sequenced fields have the same base name and are numbered sequentially starting with one (field_1, field_2, ..., field_n). They can be read into individual fields or collapsed into a single multi-dimensioned field.

Listing 4.14: Reading sequenced fields ../../../code/stk/stk_io/testsForDocumentation/setOptionToNotCollapseSequencedFields.cpp

```
TEST_F(MultipleNumberedFieldsWithSameBaseName, whenReading_collapseToSingleStkField)
{
    stk::unit_test_util::create_mesh_with__field_1__field_2__field_3(filename, get_comm());
    read_mesh(filename);
    EXPECT_EQ(1u, get_meta().get_fields(stk::topology::ELEM_RANK).size());
}

TEST_F(MultipleNumberedFieldsWithSameBaseName,
            whenReadingWitlhoutCollapseOption_threeStkFieldsAreRead)
{
    stk::unit_test_util::create_mesh_with__field_1__field_2__field_3(filename, get_comm());
    stkIo.set_option_to_not_collapse_sequenced_fields();
    read_mesh(filename);
    EXPECT_EQ(3u, get_meta().get_fields(stk::topology::ELEM_RANK).size());
}
```

4.1.13 Reading initial conditions from a field on a mesh database

This example shows how to read data from an input mesh database at a specified time and put the data into a STK Mesh field for use as initial condition data. The name of the field in the database

and the name of the STK Mesh field do not match to illustrate how to specify alternate names. The initial portion of the example, which is not shown, creates a mesh with timesteps at times 0.0, 1.0, and 2.0. The database contains a nodal field called “temp” with the same values for each node. The value is the same as the time (0.0, 1.0, and 2.0) for each time step. The example shows how to specify the reading of the field data at a specified time step.

Listing 4.15: Reading initial condition data from a mesh database ../../../code/stk/stk_io/testsForDocumentation/readInitialCondition.cpp

```
106       // ============================================================
107       //+ EXAMPLE:
108       //+ Read the value of the "temp" field at step 2 and populate
109       //+ the nodal field "temperature" for use as an initial condition
110       stk::io::StkMeshIoBroker stkIo(communicator);
111       size_t index = stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
112       stkIo.set_active_mesh(index);
113       stkIo.create_input_mesh();
114
115       stk::mesh::Field<double> &temperature = stkIo.meta_data().
116         declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK,"temperature",1);
117       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
118       stkIo.populate_bulk_data();
119
120       //+ The name of the field on the database is "temp"
121       stkIo.add_input_field(stk::io::MeshField(temperature, "temp"));
122
123       //+ Read the field values from the database at time 2.0
124       stkIo.read_defined_input_fields(2.0);
125
126       // ============================================================
127       //+ VERIFICATION
128       //+ The value of the field at all nodes should be 2.0
129       std::vector<stk::mesh::Entity> nodes;
130       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK,
131                               nodes);
132       for(size_t i=0; i<nodes.size(); i++) {
133         double *fieldDataForNode = stk::mesh::field_data(temperature, nodes[i]);
134         EXPECT_DOUBLE_EQ(2.0, *fieldDataForNode);
135       }
136
```

4.1.14 Reading initial conditions from a field on a mesh database – apply to a specified subset of mesh parts

This example is similar to the previous except that the field data read from the mesh database is limited to a subset of the parts in the model. The mesh consists of seven element blocks – one hex block and six shell blocks. The mesh database contains a single field defined on all blocks. In the example, the reading of the field is limited to the six shell element blocks; the field on the hex element block will not be initialized from the data on the mesh database. The `add_subset()` function is where this is specified.

Listing 4.16: Reading initial condition data from a mesh database ../../../code/stk/stk_io/testsForDocumentation/readInitialConditionSubset.cpp

```
63      std::string dbFieldNameShell = "ElementBlock_1";
64      std::string appFieldName = "pressure";
```

```
65
66      MPI_Comm communicator = MPI_COMM_WORLD;
67      int numProcs = stk::parallel_machine_size(communicator);
68      if (numProcs != 1) {
69        return;
70      }
71
72      {
73        // ============================================================
74        // INITIALIZATION
75        //+ Create a generated mesh containg hexes and shells with a
76        //+ single element variable -- ElementBlock_1
77        std::string input_filename = "9x9x9|shell:xyzXYZ|variables:element,1|times:1";
78
79        stk::io::StkMeshIoBroker stkIo(communicator);
80        stkIo.add_mesh_database(input_filename, "generated", stk::io::READ_MESH);
81        stkIo.create_input_mesh();
82
83        stk::mesh::MetaData &meta_data = stkIo.meta_data();
84
85        // Declare the element "pressure" field...
86        stk::mesh::Field<double> &pressure = stkIo.meta_data().
87          declare_field<stk::mesh::Field<double> >(stk::topology::ELEMENT_RANK, appFieldName,1);
88
89        // "ElementBlock_1" is the name of the element field on the input mesh.
90        stk::io::MeshField mf(pressure, dbFieldNameShell);
91
92        const stk::mesh::PartVector &all_parts = meta_data.get_mesh_parts();
93        for (size_t i=0; i < all_parts.size(); i++) {
94          const stk::mesh::Part *part = all_parts[i];
95
96          //+ Put the field on all element block parts...
97          stk::mesh::put_field(pressure, *part);
98
99          stk::topology topo = part->topology();
100         if (topo == stk::topology::SHELL_QUAD_4) {
101
102           //+ But only initialize the "pressure" field from mesh data on the shell parts.
103           mf.add_subset(*part);
104         }
105       }
106
107       stkIo.add_input_field(mf);
108       stkIo.populate_bulk_data();
109
110       double time = stkIo.get_input_io_region()->get_state_time(1);
111
112       //+ Populate the fields with data from the input mesh.
113       stkIo.read_defined_input_fields(time);
114
115
```

The previous example specified all of the subset parts on a single `MeshField`. It is also possible to specify a separate `MeshField` for each subset part. This is not the most efficient method, but can be used if other modifications of the `MeshField` are needed for each or some of the subset parts.

Listing 4.17: Reading initial condition data from a mesh database ../../../code/stk/stk_io/testsForDocumentation/readInitialConditionMultiSubset.cpp

```
73      // ============================================================
74      // INITIALIZATION
75      //+ Create a generated mesh containg hexes and shells with a
76      //+ single element variable -- pressure
```

```
77        std::string input_filename = "9x9x9|shell:xyzXYZ|variables:element,1|times:1";
78
79        stk::io::StkMeshIoBroker stkIo(communicator);
80        stkIo.add_mesh_database(input_filename, "generated", stk::io::READ_MESH);
81        stkIo.create_input_mesh();
82
83        stk::mesh::MetaData &meta_data = stkIo.meta_data();
84
85        // Declare the element "pressure" field...
86        stk::mesh::Field<double> &pressure = stkIo.meta_data().
87          declare_field<stk::mesh::Field<double> >(stk::topology::ELEMENT_RANK,appFieldName,1);
88
89        const stk::mesh::PartVector &all_parts = meta_data.get_mesh_parts();
90        for (size_t i=0; i < all_parts.size(); i++) {
91          //+ Put the field on all element block parts...
92          stk::mesh::put_field(pressure, *all_parts[i]);
93        }
94
95        // This commits BulkData and populates the coordinates, connectivity, mesh...
96        stkIo.populate_bulk_data();
97
98        double time = stkIo.get_input_io_region()->get_state_time(1);
99
100       //+ Initialize the "pressure" field from mesh data on the shell parts on demand...
101       for (size_t i=0; i < all_parts.size(); i++) {
102         stk::topology topo = all_parts[i]->topology();
103         if (topo == stk::topology::SHELL_QUAD_4) {
104
105           stk::io::MeshField mf(pressure, dbFieldNameShell);
106           mf.set_read_time(time);
107           mf.add_subset(*all_parts[i]);
108 #if 0
109           stkIo.read_input_field(mf);
110 #else
111           stkIo.add_input_field(mf);
112 #endif
113         }
114       }
115
116       //+ Populate any other fields with data from the input mesh.
117       //+ This would *not* know about the MeshFields above since
118       //+ "add_input_field()" was not called...
119       stkIo.read_defined_input_fields(time);
120
121
122
```

The final example in this section shows that the same STK field can be initialized from different database fields on different parts through the use of multiple `MeshField`s with different subsets. In this example, the “pressure” field on the shell element blocks is initialized from one database element variable and the “pressure” field on the non-shell element blocks is initialized from a different database element variable.

Listing 4.18: Reading initial condition data from a mesh database ../../../code/stk/stk_io/testsForDocumentation/readInitialConditionTwoFieldSubset.cpp

```
63    std::string dbFieldNameShell = "ElementBlock_1";
64    std::string dbFieldNameOther = "ElementBlock_2";
65    std::string appFieldName = "pressure";
66
67    MPI_Comm communicator = MPI_COMM_WORLD;
68    int numProcs = stk::parallel_machine_size(communicator);
69    if (numProcs != 1) {
70      return;
```

```
71      }
72
73      {
74        // ============================================================
75        // INITIALIZATION
76        //+ Create a generated mesh containg hexes and shells with two
77        //+ element variables -- ElementBlock_1 and ElementBlock_2
78        std::string input_filename = "9x9x9|shell:xyzXYZ|variables:element,2|times:1";
79
80        stk::io::StkMeshIoBroker stkIo(communicator);
81        stkIo.add_mesh_database(input_filename, "generated", stk::io::READ_MESH);
82        stkIo.create_input_mesh();
83
84        stk::mesh::MetaData &meta_data = stkIo.meta_data();
85
86        // Declare the element "pressure" field...
87        stk::mesh::Field<double> &pressure = stkIo.meta_data().
88          declare_field<stk::mesh::Field<double> >(stk::topology::ELEMENT_RANK, appFieldName,1);
89
90        stk::io::MeshField mf_shell(pressure, dbFieldNameShell);
91        stk::io::MeshField mf_other(pressure, dbFieldNameOther);
92
93        const stk::mesh::PartVector &all_parts = meta_data.get_mesh_parts();
94        for (size_t i=0; i < all_parts.size(); i++) {
95          const stk::mesh::Part *part = all_parts[i];
96
97          //+ Put the field on all element block parts...
98          stk::mesh::put_field(pressure, *part);
99
100         stk::topology topo = part->topology();
101         if (topo == stk::topology::SHELL_QUAD_4) {
102           //+ The shell blocks will have the pressure field initialized
103           //+ from the dbFieldNameShell database variable.
104           mf_shell.add_subset(*part);
105         }
106         else {
107           //+ The non-shell blocks will have the pressure field initialized
108           //+ from the dbFieldNameOther database variable.
109           mf_other.add_subset(*part);
110         }
111       }
112
113       stkIo.add_input_field(mf_shell);
114       stkIo.add_input_field(mf_other);
115       stkIo.populate_bulk_data();
116
117       double time = stkIo.get_input_io_region()->get_state_time(1);
118
119       //+ Populate the fields with data from the input mesh.
120       stkIo.read_defined_input_fields(time);
121
122
```

4.1.15 Reading initial conditions from a field on a mesh database – only read once

This example is the same as the previous example, except that the initial condition field will only be active for a single read. Once data has been read into the field, it is no longer active for subsequent reads. This is specified by calling `set_read_once(true)` on the input field as shown on line 125.

The `read_defined_input_fields()` function is called twice and it is verified that the field data does not change on the second call since the input field is no longer active at that call.

Listing 4.19: Reading initial condition data from a mesh database one time only ../../../code/stk/stk_io/testsForDocumentation/readInitialConditionOnce.cpp

```
106       // ============================================================
107       //+ EXAMPLE:
108       //+ Read the value of the "temp" field at step 2 and populate
109       //+ the nodal field "temperature" for use as an initial condition
110       //+ The input field should only be active for one 'read_defined_input_fields'
111       //+ call, so verify this by calling the function again at step 3 and
112       //+ then verify that the field values are still those read from step 2.
113       stk::io::StkMeshIoBroker stkIo(communicator);
114       size_t index = stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
115       stkIo.set_active_mesh(index);
116       stkIo.create_input_mesh();
117
118       stk::mesh::Field<double> &temperature = stkIo.meta_data().
119         declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK, "temperature", 1);
120       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
121       stkIo.populate_bulk_data();
122
123       //+ The name of the field on the database is "temp"
124       stk::io::MeshField input_field(temperature, "temp", stk::io::MeshField::CLOSEST);
125       input_field.set_read_once(true);
126       stkIo.add_input_field(input_field);
127
128       //+ Read the field values from the database at time 2.0
129       //+ Pass in a time of 2.2 to verify that the value returned is
130       //+ from the closest step and not interpolated.
131       stkIo.read_defined_input_fields(2.2);
132
133       // ============================================================
134       //+ VERIFICATION
135       //+ The value of the field at all nodes should be 2.0
136       std::vector<stk::mesh::Entity> nodes;
137       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK,
138                               nodes);
139       for(size_t i=0; i<nodes.size(); i++) {
140         double *fieldDataForNode = stk::mesh::field_data(temperature, nodes[i]);
141         EXPECT_DOUBLE_EQ(2.0, *fieldDataForNode);
142       }
143
144       //+ Call read_defined_input_fields again and verify that the
145       //+ input field registration is no longer active after the
146       //+ since it was specified to be "only_read_once()"
147       stkIo.read_defined_input_fields(3.0);
148
149       //+ The value of the field at all nodes should still be 2.0
150       for(size_t i=0; i<nodes.size(); i++) {
151         double *fieldDataForNode = stk::mesh::field_data(temperature, nodes[i]);
152         EXPECT_DOUBLE_EQ(2.0, *fieldDataForNode);
153       }
154
155
```

4.1.16 Reading initial conditions from a mesh database field at a specified database time

This example is similar to the previous two examples except that the database time at which the field data is to be read is specified explicitly instead of being equal to the analysis time. This is specified by calling `set_read_time()` on the input field as shown on line 141.

The `read_defined_input_fields()` function is called with an analysis time argument of `1.0`. The "flux" field gets the database field values corresponding to that time, but the "temp" field gets the database field values at the database time (2.0) time at which it is explicitly specified.

Listing 4.20: Reading initial condition data from a mesh database at a specified time ../../../code/stk/stk_io/testsForDocumentation/readInitialConditionSpecifiedTime.cpp

```
114       // ============================================================
115       //+ EXAMPLE:
116       //+ Register the reading of database fields "temp" and "flux" to
117       //+ populate the stk nodal fields "temperature" and "heat_flux"
118       //+ for use as initial conditionss.
119       //+ Specify that the "temp" field should be read from database
120       //+ time 2.0 no matter what time is specified in the read_defined_input_fields
121       //+ call.
122       //+ The "flux" field will be read at the database time corresponding
123       //+ to the analysis time passed in to read_defined_input_fields.
124
125       stk::io::StkMeshIoBroker stkIo(communicator);
126       size_t index = stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
127       stkIo.set_active_mesh(index);
128       stkIo.create_input_mesh();
129
130       stk::mesh::Field<double> &temperature = stkIo.meta_data().
131         declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK, "temperature", 1);
132       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
133
134       stk::mesh::Field<double> &heat_flux = stkIo.meta_data().
135         declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK, "heat_flux", 1);
136       stk::mesh::put_field(heat_flux, stkIo.meta_data().universal_part());
137       stkIo.populate_bulk_data();
138
139       // The name of the field on the database is "temp"
140       stk::io::MeshField temp_field(temperature, "temp", stk::io::MeshField::CLOSEST);
141       temp_field.set_read_time(2.0);
142       stkIo.add_input_field(temp_field);
143
144       // The name of the field on the database is "flux"
145       stk::io::MeshField flux_field(heat_flux, "flux", stk::io::MeshField::CLOSEST);
146       stkIo.add_input_field(flux_field);
147
148       //+ Read the field values from the database at time 1.0
149       //+ The value of "flux" will be the values from database time 1.0
150       //+ However, the value of "temp" will be the values from database time 2.0
151       stkIo.read_defined_input_fields(1.0);
152
153       // ============================================================
154       //+ VERIFICATION
155       std::vector<stk::mesh::Entity> nodes;
156       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK,
157                               nodes);
158
159       //+ The value of the "temperature" field at all nodes should be 2.0
160       for(size_t i=0; i<nodes.size(); i++) {
161         double *fieldDataForNode = stk::mesh::field_data(temperature, nodes[i]);
```

```
162         EXPECT_DOUBLE_EQ(2.0, *fieldDataForNode);
163       }
164
165       //+ The value of the "heat_flux" field at all nodes should be 1.0
166       for(size_t i=0; i<nodes.size(); i++) {
167         double *fieldDataForNode = stk::mesh::field_data(heat_flux, nodes[i]);
168         EXPECT_DOUBLE_EQ(1.0, *fieldDataForNode);
169       }
170
```

4.1.17 Reading field data from a mesh database – interpolating between database times

This example shows how to read data from an input mesh database at multiple times. The database field values are linearly interpolated if the analysis time does not match an existing database time. The initial portion of the example, which is not shown, creates a mesh with time steps at times 0.0, 1.0, and 2.0. The database contains a nodal field called "temp" with the same values for each node. The value is the same as the time (0.0, 1.0, and 2.0) for each time step. The example shows how to specify the reading of the field data at multiple steps and linearly interpolating the database data to the specified analysis times. Line 128 shows how to specify that the field data are to be linear interpolated.

Listing 4.21: Linearly interpolating field data from a mesh database ../../../code/stk/stk_io/testsForDocumentation/interpolateNodalField.cpp

```
106       // ============================================================
107       //+ EXAMPLE:
108       //+ The input mesh database has 3 timesteps with times 0.0, 1.0, 2.0,
109       //+ The value of the field "temp" is equal to the time
110       //+ Read the "temp" value at times 0.0 to 2.0 with an interval
111       //+ of 0.1 (0.0, 0.1, 0.2, 0.3, ..., 2.0) and verify that
112       //+ the field contains the correct interpolated value.
113       stk::io::StkMeshIoBroker stkIo(communicator);
114       stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
115       stkIo.create_input_mesh();
116
117       stk::mesh::Field<double> &temperature = stkIo.meta_data().
118         declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK,"temperature",1);
119       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
120
121       stkIo.populate_bulk_data();
122
123       std::vector<stk::mesh::Entity> nodes;
124       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK, nodes);
125
126       //+ Specify that the field data are to be linear interpolated.
127       stkIo.add_input_field(stk::io::MeshField(temperature, "temp",
128                                                stk::io::MeshField::LINEAR_INTERPOLATION));
129
130       //+ If the same stk field (temperature) is added more than once,
131       //+ the first database name and settings will be used.  For example,
132       //+ the add_input_field below will be ignored with no error or warning.
133       stkIo.add_input_field(stk::io::MeshField(temperature, "temp-again",
134                                                stk::io::MeshField::LINEAR_INTERPOLATION));
135
136       for (size_t i=0; i < 21; i++) {
137         double time = i/10.0;
```

```
138         //+ Read the field values from the database and verify that they
139         //+ are interpolated correctly.
140         stkIo.read_defined_input_fields(time);
141
142         // ============================================================
143         //+ VERIFICATION
144         // The value of the "temperature" field at all nodes should be 'time'
145         for(size_t j=0; j<nodes.size(); j++) {
146           double *fieldData = stk::mesh::field_data(temperature, nodes[j]);
147           EXPECT_DOUBLE_EQ(time, *fieldData);
148         }
149       }
150
```

4.1.18 Combining restart and interpolation of field data

This example shows how to specify that an analysis, that is using field interpolation, should be restarted. This requires two input databases: one that contains the restart data and another that contains the field data to be interpolated.

The initial portion of the example, which is not shown, creates a restart database with several nodal and element fields containing three time steps at times 0.0, 1.0, and 2.0. It then also creates a database containing the field values which will be interpolated. This database contains 10 time steps (0.0 to 9.0) with the nodal field "temp". The value of the field at each time step is equal to the database time (0.0 to 9.0).

The `add_mesh_database()` function is called twice – once for each database. Since there are multiple mesh databases, the `set_active_mesh()` function is called to specify which mesh is active for subsequent calls. The fields that are to be read from each database are specified using `add_all_mesh_fields_as_input_fields()` for the restart database and `add_input_field()` for the interpolated field database. Note that the file index for the interpolated field database is passed to the `add_input_field()` since that database is not active at the time of the call.

The example then "restarts" the analysis by setting the restart database as the *active mesh* and reads the restart field data at time 1.0. The active mesh is then switched to the mesh database containing the "temp" field and the analysis is then continued up to time 9.0 with the values for the temperature field being interpolated.

Listing 4.22: Combining restart and field interpolation
../../../code/stk/stk_io/testsForDocumentation/restartInterpolatedField.cpp

```
142       // ============================================================
143       //+ EXAMPLE:
144       //+ The restart mesh database has 3 timesteps with times 0.0, 1.0, 2.0,
145       //+ and several fields.
146       //+
147       //+ The initial condition database has 10 timesteps with times
148       //+ 0.0, 1.0, ..., 9.0 and a nodal variable "temp"
149       //+ The value of the field "temp" is equal to the time
150       //+
151       //+ The example will read the restart database at time 1.0
```

```
152       //+ and then simulate continuing the analysis at that time
153       //+ reading the initial condition data from the other database
154       //+ interpolating this data.
155       stk::io::StkMeshIoBroker stkIo(communicator);
156       size_t ic = stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
157       size_t rs = stkIo.add_mesh_database(rs_name, stk::io::READ_RESTART);
158
159       //+ "Restart" the calculation...
160       double time = 1.0;
161       stkIo.set_active_mesh(rs);
162       stkIo.create_input_mesh();
163
164       stkIo.add_all_mesh_fields_as_input_fields();
165
166       stk::mesh::Field<double> &temperature = stkIo.meta_data().
167         declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK, "temperature", 1);
168       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
169
170       // The name of the field on the initial condition database is "temp"
171       stkIo.add_input_field(ic, stk::io::MeshField(temperature, "temp",
172                                                    stk::io::MeshField::LINEAR_INTERPOLATION));
173       stkIo.populate_bulk_data();
174
175       std::vector<stk::mesh::Entity> nodes;
176       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK, nodes);
177
178       //+ Read restart data
179       stkIo.read_defined_input_fields(time);
180
181       //+ Switch active mesh to "initial condition" database
182       stkIo.set_active_mesh(ic);
183
184       double delta_time = 1.0 / 4.0;
185       while (time <= 9.0) {
186         //+ Read the field values from the database and verify that they
187         //+ are interpolated correctly.
188         stkIo.read_defined_input_fields(time);
189
190         // ============================================================
191         //+ VERIFICATION
192         // The value of the "temperature" field at all nodes should be 'time'
193         for(size_t i=0; i<nodes.size(); i++) {
194           double *fieldDataForNode = stk::mesh::field_data(temperature, nodes[i]);
195           EXPECT_DOUBLE_EQ(time, *fieldDataForNode);
196         }
197         time += delta_time;
198       }
199
```

4.1.19 Interpolating field data from a mesh database with only a single database time

If an application specifies that the mesh database field data should be linearly interpolated, but the mesh database only has a single time step, then the field data will not be interpolated and instead, the values read from that single time will be used.

The initial portion of the example, which is not shown, creates a mesh with a time step at time 1.0. The database contains a nodal field called “temp” with the same values for each node. The value is the same as the time (1.0).

The example specifies that the field data should be linearly interpolated and then reads the data at multiple steps. Since there is only a single step on the mesh database, all field values are equal to the database values at that step.

Listing 4.23: Linearly interpolating field data from a mesh database with only a single step ../../../code/stk/stk_io/testsForDocumentation/interpolateSingleStep.cpp

```
102       // ============================================================
103       //+ EXAMPLE:
104       //+ The input mesh database has 1 timesteps with time 1.0
105       //+ The value of the field "temp" is equal to the time
106       //+ Read the "temp" value at times 0.0 to 2.0 with an interval
107       //+ of 0.1 (0.0, 0.1, 0.2, 0.3, ..., 2.0) and verify that
108       //+ the field value does not change since there are not
109       //+ enough steps to do any interpolation.
110       //+
111       stk::io::StkMeshIoBroker stkIo(communicator);
112       stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
113       stkIo.create_input_mesh();
114
115       stk::mesh::Field<double> &temperature = stkIo.meta_data().
116         declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK,"temperature",1);
117       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
118
119       // The name of the field on the database is "temp"
120       stkIo.add_input_field(stk::io::MeshField(temperature, "temp",
121                                                stk::io::MeshField::LINEAR_INTERPOLATION));
122
123       stkIo.populate_bulk_data();
124
125       std::vector<stk::mesh::Entity> nodes;
126       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK, nodes);
127
128       for (size_t i=0; i < 21; i++) {
129         double time = i/10.0;
130         //+ Read the field values from the database and verify that they
131         //+ are interpolated correctly.
132         stkIo.read_defined_input_fields(time);
133
134         // ============================================================
135         //+ VERIFICATION
136         // The value of the "temperature" field at all nodes should be 1.0
137         for(size_t j=0; j<nodes.size(); j++) {
138           double *fieldData = stk::mesh::field_data(temperature, nodes[j]);
139           EXPECT_DOUBLE_EQ(1.0, *fieldData);
140         }
141       }
142
```

4.1.20 Interpolating field data from a mesh database when time is outside database time interval

If an application specifies that the mesh database field data should be linearly interpolated, but requests data at times outside the interval of times present on the mesh database, then the values at the closest database time will be used instead. In other words, the database values are not extrapolated.

The initial portion of the example, which is not shown, creates a mesh with two time steps at times

1.0 and 2.0. The database contains a nodal field called “temp” with the same values for each node. The value is the same as the time (1.0 or 2.0).

The example specifies that the field data should be linearly interpolated and then reads the data at multiple times from 0.0 to 3.0. Since the database only contains data at times 1.0 and 2.0, the field values at times 0.0 to 1.0 will be set to the database values at time 1.0 and the field values at times 2.0 to 3.0 will be set to the database values at time 2.0. The field values at times 1.0 to 2.0 will be linearly interpolated from the database values.

Listing 4.24: Linearly interpolating field data when the time is outside the database time interval ../../../code/stk/stk_io/testsForDocumentation/interpolateOutsideRange.cpp

```
104       // ============================================================
105       //+ EXAMPLE:
106       //+ The input mesh database has 2 timesteps with time 1.0 and 2.0
107       //+ The value of the field "temp" is equal to the time
108       //+ Read the "temp" value at times 0.0 to 3.0 with an interval
109       //+ of 0.1 (0.0, 0.1, 0.2, 0.3, ..., 2.0).
110       //+
111       //+ The times 0.0 to 1.0 and 2.0 to 3.0 are outside
112       //+ the range of the mesh database so no interpolation
113       //+ or extrapolation will occur -- the field values
114       //+ will be set to the values at the nearest time.
115       //+
116       //+ Verify that the values from times 0.0 to 1.0
117       //+ are equal to 1.0 and that the values from 2.0 to 3.0
118       //+ are equal to 2.0.
119       //+ The field values from 1.0 to 2.0 will be interpolated
120       //+
121       stk::io::StkMeshIoBroker stkIo(communicator);
122       stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
123       stkIo.create_input_mesh();
124
125       stk::mesh::Field<double> &temperature = stkIo.meta_data().
126         declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK,"temperature",1);
127       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
128
129       stkIo.populate_bulk_data();
130
131       std::vector<stk::mesh::Entity> nodes;
132       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK, nodes);
133
134       // The name of the field on the database is "temp"
135       stkIo.add_input_field(stk::io::MeshField(temperature, "temp",
136                                                stk::io::MeshField::LINEAR_INTERPOLATION));
137
138       for (size_t i=0; i < 21; i++) {
139         double time = i/10.0;
140         //+ Read the field values from the database and verify that they
141         //+ are interpolated correctly.
142         stkIo.read_defined_input_fields(time);
143
144         // ============================================================
145         //+ VERIFICATION
146
147         double expected_value = time;
148         if (time <= 1.0)
149           expected_value = 1.0;
150         if (time >= 2.0)
151           expected_value = 2.0;
152
153         for(size_t j=0; j<nodes.size(); j++) {
154           double *fieldData = stk::mesh::field_data(temperature, nodes[j]);
155           EXPECT_DOUBLE_EQ(expected_value, *fieldData);
```

```
156           }
157         }
158
```

4.1.21 Error condition – reading initial conditions from a field that does not exist on a mesh database

This example shows the behavior when the application specifies that initial condition or restart data should be read from the input database, but one or more of the specified fields do not exist on the database. The application specifies that the data for the field “displacement” is to be populated from the database field “disp”, which does not exist. Two scenarios are possible. In the first, the application passes in a vector which on return from the `read_defined_input_fields()` function will contain a list of all fields that were not found, with one entry for each missing field state. In the second, the vector is omitted in the call to `read_defined_input_fields();` in this case, the code will print an error message and throw an exception if there are any fields not found.

Listing 4.25: Specifying initial conditions from a non-existent field ../../../code/stk/stk_io/testsForDocumentation/handleMissingFieldOnRead.cpp

```
108       // ============================================================
109       //+ EXAMPLE:
110       //+ Demonstrate what happens when application requests the
111       //+ reading of a field that does not exist on the input
112       //+ mesh database.  The nodal field "displacement" is
113       //+ requested for input from the database field "disp" which
114       //+ does not exist.
115       stk::io::StkMeshIoBroker stkIo(communicator);
116       size_t index = stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
117       stkIo.set_active_mesh(index);
118       stkIo.create_input_mesh();
119
120       stk::mesh::Field<double> &temperature =
121           stkIo.meta_data().declare_field<stk::mesh::Field<double> >(
122                                              stk::topology::NODE_RANK,"temperature",1);
123       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
124
125       stk::mesh::Field<double> &displacement =
126           stkIo.meta_data().declare_field<stk::mesh::Field<double> >(
127                                            stk::topology::NODE_RANK,"displacement",3);
128       stk::mesh::put_field(displacement, stkIo.meta_data().universal_part());
129       stkIo.populate_bulk_data();
130
131       // The name of the field on the database is "temp"
132       // This field does exist and should be read correctly
133       stkIo.add_input_field(stk::io::MeshField(temperature, "temp"));
134
135       //+ The name of the field on the database is "disp"
136       //+ This field does not exist and will not be found.
137       stkIo.add_input_field(stk::io::MeshField(displacement, "disp"));
138
139
140       //+ Read the field values from the database at time 2.0
141       //+ The 'missing_fields' vector will contain the names of
142       //+ any fields that were not found.
143       std::vector<stk::io::MeshField> missing_fields;
144       stkIo.read_defined_input_fields(2.0, &missing_fields);
```

```
145
146       // =============================================================
147       //+ VERIFICATION
148       //+ The 'missing' vector should be of size 1 and contain
149       //+ 'disp'
150       EXPECT_EQ(2u, missing_fields.size());
151       EXPECT_EQ("disp", missing_fields[0].db_name());
152       EXPECT_EQ("displacement", missing_fields[0].field()->name());
153       EXPECT_EQ("disp", missing_fields[1].db_name());
154       EXPECT_EQ("displacement_STKFS_N", missing_fields[1].field()->name());
155
156       // The value of the "temperature" field at all nodes should be 2.0
157       std::vector<stk::mesh::Entity> nodes;
158       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK,
159                               nodes);
160       for(size_t i=0; i<nodes.size(); i++) {
161         double *fieldDataForNode =
162                 stk::mesh::field_data(temperature, nodes[i]);
163         EXPECT_DOUBLE_EQ(2.0, *fieldDataForNode);
164       }
165
```

This example is the same as the previous except that instead of passing in the vector to hold the missing fields, the application will throw an exception for the missing field. Note that if the application throws an exception, it will not read any field data even for the fields that do exist.

Listing 4.26: Specifying initial conditions from a non-existent field
../../../code/stk/stk_io/testsForDocumentation/handleMissingFieldOnReadThrow.cpp

```
136       //+ If read the fields, but don't pass in the 'missing_fields'
137       //+ vector, the code will print an error message and throw an
138       //+ exception if it doesn't find all of the requested fields.
139       EXPECT_ANY_THROW(stkIo.read_defined_input_fields(2.0));
140
141       //+ If code throws due to missing field(s), it will NOT read
142       //+ even the fields that exist.
143
```

4.1.22 Interpolation of fields on database with negative times

Although it is not common, there are occasions when an analysis will use negative times. For example, an analysis may run from time -3.0 to 0.0 to "preload" a structure and then continue from time 0.0 onward to analyze the preloaded structure. This example shows that the field interpolation capability works correctly when the mesh database and the analysis use negative times.

Listing 4.27: Interpolating fields on a database with negative times
../../../code/stk/stk_io/testsForDocumentation/interpolateFieldNegativeTime.cpp

```
107       // =============================================================
108       //+ EXAMPLE:
109       //+ The input mesh database has 3 timesteps with times -2.0, -1.0, 0.0.
110       //+ The value of the field "temp" is equal to the time
111       //+ Read the "temp" value at times -2.0 to 0.0 with an interval
112       //+ of 0.1 (-2.0, -1.9, -1.8, ..., 0.0) and verify that
113       //+ the field contains the correct interpolated value.
114       stk::io::StkMeshIoBroker stkIo(communicator);
115       stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
```

```
116       stkIo.create_input_mesh();
117
118       stk::mesh::Field<double> &temperature = stkIo.meta_data().
119         declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK, "temperature", 1);
120       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
121
122       stkIo.populate_bulk_data();
123
124       std::vector<stk::mesh::Entity> nodes;
125       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK, nodes);
126
127       // The name of the field on the database is "temp"
128       stkIo.add_input_field(stk::io::MeshField(temperature, "temp",
129                                                stk::io::MeshField::LINEAR_INTERPOLATION));
130
131       for (int i=-20; i <= 0; i++) {
132         double time = i/10.0;
133         //+ Read the field values from the database and verify that they
134         //+ are interpolated correctly.
135         stkIo.read_defined_input_fields(time);
136
137         // ============================================================
138         //+ VERIFICATION
139         // The value of the "temperature" field at all nodes should be 'time'
140         for(size_t j=0; j<nodes.size(); j++) {
141           double *fieldData = stk::mesh::field_data(temperature, nodes[j]);
142           EXPECT_DOUBLE_EQ(time, *fieldData);
143         }
144       }
145
```

4.1.23 Interpolation of fields on database with non-monotonically increasing times

In some cases, the database from which the field values are being interpolated may contain non-monotonically increasing time values. For example, the time steps could contain the values 2.0 at step 1, 0.0 at step 2, and 1.0 at step 3. The example shows that the field interpolation capability works correctly in this case.

Listing 4.28: Interpolating fields on a database with non-monotonically increasing times ../../../code/stk/stk_io/testsForDocumentation/interpolateFieldNonMonotonicTime.cpp

```
107       // ============================================================
108       //+ EXAMPLE:
109       //+ The input mesh database has 3 timesteps with times 2.0, 0.0, 1.0
110       //+ which are non-monotonically increasing.
111       //+ The value of the field "temp" is equal to the time
112       //+ Read the "temp" value at times 0.0 to 2.0 with an interval
113       //+ of 0.1 (0.0, 0.1, 0.2, ..., 2.0) and verify that
114       //+ the field contains the correct interpolated value.
115       stk::io::StkMeshIoBroker stkIo(communicator);
116       stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
117       stkIo.create_input_mesh();
118
119       stk::mesh::Field<double> &temperature = stkIo.meta_data().
120         declare_field<stk::mesh::Field<double> >(stk::topology::NODE_RANK, "temperature", 1);
121       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
122
123       stkIo.populate_bulk_data();
```

```
124
125       std::vector<stk::mesh::Entity> nodes;
126       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK, nodes);
127
128       // The name of the field on the database is "temp"
129       stkIo.add_input_field(stk::io::MeshField(temperature, "temp",
130                                                stk::io::MeshField::LINEAR_INTERPOLATION));
131
132       for (int i=0; i < 21; i++) {
133         double time = i/10.0;
134         //+ Read the field values from the database and verify that they
135         //+ are interpolated correctly.
136         stkIo.read_defined_input_fields(time);
137
138         // ============================================================
139         //+ VERIFICATION
140         // The value of the "temperature" field at all nodes should be 'time'
141         for(size_t j=0; j<nodes.size(); j++) {
142           double *fieldData = stk::mesh::field_data(temperature, nodes[j]);
143           EXPECT_DOUBLE_EQ(time, *fieldData);
144         }
145       }
146
```

4.1.24 Arbitrary analysis time to database time mapping during field input

There are instances in which the analysis times do not exactly correspond to the times on the mesh database. An example is a mesh database with times in microseconds and the analysis using seconds for the time units. Another example is when the conditions specified on the mesh database describe a cyclic loading over a small time period, but the analysis time runs over multiples of this period.

The `InputFile` class in STK Mesh IO Broker module contains several options for mapping the analysis time to the database time. These include: offset, scale, period, startup, period type, start time, and stop time.

To describe the mapping from analysis time to database time we will use the following notation:

- a variable of type t_x is in units of time.
- t_{app} is application time.
- t_{db} is database time, which is the time that will be used to query the database.
- t_{period} is the length of the cyclic period; it is 0.0 if not cyclic.
- *scale* is the time scaling factor.
- t_{offset} is the time offset.
- The cyclic behavior can either by specified as *CYCLIC* or *REVERSING*. In the cyclic case, the time would repeat as 1,2,3,1,2,3,...; the reversing case would repeat as 1,2,3,2,1,2,3,,.... Both of these have a t_{period} of length 2.

We now describe the mapping:

- If: $t_{app} < t_{start}$ or $t_{app} > t_{stop}$ Then the field is inactive.
- If: $t_{app} < t_{startup}$ Then $t_{db} = t_{app}$.
- Else if cyclic behavior is *CYCLIC* Then $t_{db} = t_{startup} + \mod(t_{app} - t_{startup}, t_{period})$.
- Else if cyclic behavior is *REVERSING* Then
  - Let $t_{pm} = \mod(t_{app} - t_{startup}, 2 \times t_{period})$
  - If: $(t_{pm} \leq t_{period})$ Then $t_{db} = t_{startup} + t_{pm}$
  - Else: $t_{db} = t_{startup} + (2 \times t_{period} - t_{pm})$.
- Finally: $t_{db} = t_{db} \times scale + t_{offset}$.

The example below shows an input mesh database containing a nodal field named "temp". The database contains 3 steps with times 0.0, 10.0, and 20.0; the value of the field at each time is equal to the time value (0.0, 10.0, or 20.0).

The analysis wants to use the data on this mesh to provide linearly interpolated values for the analysis field "temperature". The mesh database values will be defined as `REVERSING` cyclic with a period length of 2.0; in addition, the times will be scaled by 10. This should result in a mapping of application time (t_{app}) to database time (t_{db}) of:

t_{app}	0	1	2	3	4	5	6	7	8	9	10
t_{db}	0	10	20	10	0	10	20	10	0	10	20

Listing 4.29: Arbitrary analysis time to database time mapping during field input ../../../code/stk/stk_io/testsForDocumentation/interpolateFieldCyclic.cpp

```
107       // ============================================================
108       //+ EXAMPLE:
109       //+ The input mesh database has 3 timesteps with times 0.0, 10.0, 20.0,
110       //+ The value of the field "temp" is equal to the time
111       //+ Read the "temp" value at times 0.0 to 10.0 with an interval
112       //+ of 0.25 (0.0, 0.25, 0.50, 0.75, ..., 10.0)
113       //+ The mapping from analysis time (0.0 to 10.0) to database
114       //+ time will be reverse cyclic and scaled.
115       //+
116       //+ The parameters are:
117       //+ * period = 2.0
118       //+ * scale = 10.0
119       //+ * offset = 0.0
120       //+ * cycle type = REVERSING
121       //+
122       //+ Analysis Time and DB_Time:
123       //+ 0  1  2  3   4   5   6   7   8   9  10
124       //+ 0 10 20 10   0  10  20  10   0  10  20
125       //+
126
127       stk::io::StkMeshIoBroker stkIo(communicator);
128       size_t idx = stkIo.add_mesh_database(ic_name, stk::io::READ_MESH);
129       stkIo.create_input_mesh();
130
131       stk::mesh::Field<double> &temperature = stkIo.
132         meta_data().declare_field<stk::mesh::Field<double> >
```

```
133         (stk::topology::NODE_RANK, "temperature", 1);
134       stk::mesh::put_field(temperature, stkIo.meta_data().universal_part());
135
136       stkIo.populate_bulk_data();
137
138       std::vector<stk::mesh::Entity> nodes;
139       stk::mesh::get_entities(stkIo.bulk_data(), stk::topology::NODE_RANK, nodes);
140
141       // The name of the field on the database is "temp"
142       stkIo.add_input_field(stk::io::MeshField(temperature, "temp",
143                                                stk::io::MeshField::LINEAR_INTERPOLATION));
144
145       //+ Set the periodic parameters on the input mesh...
146       double period_length = 2.0;
147       double startup = 0.0;
148       double scale = 10.0;
149       stkIo.get_mesh_database(idx)
150         .set_periodic_time(period_length, startup, stk::io::InputFile::REVERSING)
151         .set_scale_time(scale)
152         .set_start_time(0.0).set_offset_time(0.0).set_stop_time(999.0); // These are optional
153       double delta_time = 0.25;
154       double time = 0.0;
155       double expected = 0.0;
156       double exp_inc = 10.0 * delta_time;
157
158       while (time <= 10.0) {
159
160         //+ Read the field values from the database and verify that they
161         //+ are interpolated correctly.
162         stkIo.read_defined_input_fields(time);
163
164         // ============================================================
165         //+ VERIFICATION
166         // The value of the "temperature" field at all nodes should be 'expected'
167         for(size_t i=0; i<nodes.size(); i++) {
168           double *fieldData = stk::mesh::field_data(temperature, nodes[i]);
169           EXPECT_DOUBLE_EQ(expected, *fieldData);
170         }
171         time += delta_time;
172         expected += exp_inc;
173         if (expected >= 20.0 || expected <= 0.0) {
174           exp_inc = -exp_inc;
175         }
176       }
177
```

4.1.25 Error condition – specifying interpolation for an integer field

This example shows the behavior when the application specifies that linear interpolation should be used for an integer field. Although there are a few instances in which this could be valid, it is not supported and an exception will be thrown when the field is registered.

Listing 4.30: Error condition – specifying interpolation of an integer field
../../../code/stk/stk_io/testsForDocumentation/interpolateIntegerFieldInvalid.cpp

```
58       // ============================================================
59       //+ EXAMPLE:
60       //+ Interpolated fields cannot be of type integer.
61       //+ An exception will be thrown if you try to register an
62       //+ integer interpolated field.
63
```

```
64        stk::io::StkMeshIoBroker stkIo(communicator);
65
66        const std::string generatedFileName = "generated:8x8x8|nodeset:xyz";
67        stkIo.add_mesh_database(generatedFileName, stk::io::READ_MESH);
68        stkIo.create_input_mesh();
69
70        stk::mesh::Field<int> &integer_field = stkIo.meta_data().
71          declare_field<stk::mesh::Field<int> >(stk::topology::NODE_RANK, "int_field", 1);
72        stk::mesh::put_field(integer_field, stkIo.meta_data().universal_part());
73        stkIo.populate_bulk_data();
74
75        EXPECT_ANY_THROW(stkIo.add_input_field(stk::io::MeshField(integer_field,
76                                                                  "int_field",
77                                       stk::io::MeshField::LINEAR_INTERPOLATION)));
78
```

4.1.26 Working with element attributes

Listing 4.31: Working with element attributes
../../../code/stk/stk_io/testsForDocumentation/readAttributes.cpp

```
57 std::vector<double> get_attributes_of_first_element(const stk::mesh::BulkData &bulk, const
               stk::mesh::Part *ioPart)
58 {
59     stk::mesh::FieldVector attributeFields =
               stk::io::get_attribute_fields_for_part(bulk.mesh_meta_data(), ioPart);
60
61     stk::mesh::EntityVector elements;
62     stk::mesh::get_selected_entities(*ioPart, bulk.buckets(stk::topology::ELEM_RANK),
               elements);
63
64     std::vector<double> attributes;
65     if(!elements.empty())
66     {
67         for(const stk::mesh::FieldBase *field : attributeFields)
68         {
69             unsigned numAttribute = stk::mesh::field_scalars_per_entity(*field, elements[0]);
70             double *dataForElement = static_cast<double*> (stk::mesh::field_data(*field,
               elements[0]));
71             for(unsigned i=0; i<numAttribute; ++i)
72                 attributes.push_back(dataForElement[i]);
73         }
74     }
75     return attributes;
76 }
77
78 TEST_F(ExodusFileWithAttributes, readAttributes_haveFieldsWithAttributes)
79 {
80     setup_mesh("hex_spider.exo", stk::mesh::BulkData::AUTO_AURA);
81
82     const stk::mesh::Part *partBlock2  = get_meta().get_part("block_2");
83     const stk::mesh::Part *partBlock10 = get_meta().get_part("block_10");
84
85     EXPECT_EQ(1u, get_attributes_of_first_element(get_bulk(), partBlock2).size());
86     EXPECT_EQ(7u, get_attributes_of_first_element(get_bulk(), partBlock10).size());
87 }
88
89 TEST_F(ExodusFileWithAttributes, addAttribute_haveFieldsWithAttribute)
90 {
91     allocate_bulk(stk::mesh::BulkData::AUTO_AURA);
92
93     stk::io::StkMeshIoBroker stkIo;
```

```
94      stkIo.set_bulk_data(get_bulk());
95      stkIo.add_mesh_database("hex_spider.exo", stk::io::READ_MESH);
96      stkIo.create_input_mesh();
97
98      double initialValue = 0.0;
99      auto &newAttrField =
                get_meta().declare_field<stk::mesh::Field<double>>(stk::topology::ELEM_RANK,
                "newAttr");
100     stk::io::mark_field_as_attribute(newAttrField);
101     const stk::mesh::Part *partBlock10 = get_meta().get_part("block_10");
102     stk::mesh::put_field(newAttrField, *partBlock10, &initialValue);
103
104     stkIo.populate_bulk_data();
105
106     EXPECT_EQ(8u, get_attributes_of_first_element(get_bulk(), partBlock10).size());
107 }
```

4.1.27 Create an output mesh with a subset of the mesh parts

If a results file that only contains a portion or subset of the parts existing in the STK Mesh is wanted, this can be specified by creating a `Selector` (see Section 1.4) containing the desired output parts and then calling the `set_subset_selector()` function with that `Selector` as an argument. This is illustrated in the following example.

Listing 4.32: Creating output mesh containing a subset of the mesh parts
../../../code/stk/stk_io/testsForDocumentation/subsettingOutputDB.cpp

```
67      // ============================================================
68      // INITIALIZATION
69      std::string s_elems_per_edge = Ioss::Utils::to_string(num_elems_per_edge);
70
71      //+ Create a generated mesh containg hexes and shells.
72      std::string input_filename = s_elems_per_edge + "x" +
73                                   s_elems_per_edge + "x" +
74                                   s_elems_per_edge + "|shell:xyzXYZ";
75
76      stk::io::StkMeshIoBroker stkIo(communicator);
77      size_t index = stkIo.add_mesh_database(input_filename, "generated",
78                                             stk::io::READ_MESH);
79      stkIo.set_active_mesh(index);
80      stkIo.create_input_mesh();
81      stkIo.populate_bulk_data();
82
83      stk::mesh::MetaData &meta_data = stkIo.meta_data();
84      const stk::mesh::PartVector &all_parts = meta_data.get_mesh_parts();
85
86      // ============================================================
87      //+ EXAMPLE
88      //+ Create a selector containing just the shell parts.
89      stk::mesh::Selector shell_subset;
90      for (size_t i=0; i < all_parts.size(); i++) {
91        const stk::mesh::Part *part = all_parts[i];
92        stk::topology topo = part->topology();
93        if (topo == stk::topology::SHELL_QUAD_4) {
94          shell_subset |= *part;
95        }
96      }
97
98      // Create the output...
99      size_t fh = stkIo.create_output_mesh(resultsFilename,
100                                           stk::io::WRITE_RESULTS);
101
```

```
102       //+ Specify that only the subset of parts selected by the
103       //+ "shell_subset" selector will be on the output database.
104       stkIo.set_subset_selector(fh, shell_subset);
105       stkIo.write_output_mesh(fh);
106       // Verification omitted...
107
```

4.1.28 Writing and reading global variables

The following example shows the use of global variables for a scalar double precision floating point value, but a similar interface exists for working with vectors of global values. The example also shows two methods for handling the error condition of accessing a nonexistent global variable.

Listing 4.33: Writing and reading a global variable
../../../code/stk/stk_io/testsForDocumentation/writingAndReadingGlobalVariables.cpp

```
48 TEST(StkMeshIoBrokerHowTo, writeAndReadGlobalVariables)
49 {
50     const std::string restartFileName = "OneGlobalDouble.restart";
51     const std::string timeStepVarName = "timeStep";
52     const double timeStepSize = 1e-6;
53     const double currentTime = 1.0;
54     MPI_Comm communicator = MPI_COMM_WORLD;
55     int numProcs = stk::parallel_machine_size(communicator);
56     if (numProcs != 1) {
57       return;
58     }
59
60     //+ Write restart file with time step size as a global variable
61     {
62         stk::io::StkMeshIoBroker stkIo(communicator);
63         const std::string exodusFileName = "generated:1x1x8";
64         stkIo.add_mesh_database(exodusFileName, stk::io::READ_MESH);
65         stkIo.create_input_mesh();
66         stkIo.populate_bulk_data();
67
68         size_t fileIndex =
69           stkIo.create_output_mesh(restartFileName, stk::io::WRITE_RESTART);
70         stkIo.add_global(fileIndex, timeStepVarName, Ioss::Field::REAL);
71         stkIo.begin_output_step(fileIndex, currentTime);
72         stkIo.write_global(fileIndex, timeStepVarName, timeStepSize);
73         stkIo.end_output_step(fileIndex);
74     }
75
76     //+ Read restart file with time step size as a global variable
77     {
78         stk::io::StkMeshIoBroker stkIo(communicator);
79         stkIo.add_mesh_database(restartFileName, stk::io::READ_RESTART);
80         stkIo.create_input_mesh();
81         stkIo.populate_bulk_data();
82         stkIo.read_defined_input_fields(currentTime);
83         std::vector<std::string> globalNamesOnFile;
84         stkIo.get_global_variable_names(globalNamesOnFile);
85
86         ASSERT_EQ(1u, globalNamesOnFile.size());
87         EXPECT_STRCASEEQ(timeStepVarName.c_str(),
88                          globalNamesOnFile[0].c_str());
89         double timeStepSizeReadFromFile = 0.0;
90         stkIo.get_global(globalNamesOnFile[0], timeStepSizeReadFromFile);
91         const double tolerance = 1e-16;
92         EXPECT_NEAR(timeStepSize, timeStepSizeReadFromFile, tolerance);
```

```
93
94          //+ If try to get a global that does not exist, will throw
95          //+ an exception by default...
96          double value = 0.0;
97          EXPECT_ANY_THROW(stkIo.get_global("does_not_exist", value));
98
99          //+ If the application wants to handle the error instead (without a try/catch),
100         //+ can pass in an optional boolean:
101         bool abort_if_not_found = false;
102         bool found = stkIo.get_global("does_not_exist", value, abort_if_not_found);
103         ASSERT_FALSE(found);
104     }
105
106     unlink(restartFileName.c_str());
107 }
```

4.1.29 Writing and reading global parameters

The following example shows the use of `stk::util::Parameter` objects for global variable output and input. The example defines several parameters of type double, integer, vector of doubles, and a vector of integers. The list containing these parameters is iterated and each is defined to be an output global variable. Then, each variable is written in the time step loop. At the end of writing, the file is reopened for reading and the parameter values are restored and checked to make sure the correct values were read.

Listing 4.34: Writing and reading parameters as global variables ../../../code/stk/stk_io/testsForDocumentation/writingAndReadingGlobalParameters.cpp

```
49  TEST(StkMeshIoBrokerHowTo, writeAndReadGlobalParameters)
50  {
51    // ============================================================
52    //+ INITIALIZATION
53    const std::string file_name = "GlobalParameters.e";
54    MPI_Comm communicator = MPI_COMM_WORLD;
55
56    // Add some parameters to write and read...
57    stk::util::ParameterList params;
58    params.set_param("PI", 3.14159);  // Double
59    params.set_param("Answer", 42);   // Integer
60
61    std::vector<double> my_vector = { 2.78, 5.30, 6.21 };
62    params.set_param("doubles", my_vector); // Vector of doubles...
63
64    std::vector<int> ages = { 55, 49, 21, 19};
65    params.set_param("Ages", ages);   // Vector of integers...
66
67    {
68      stk::io::StkMeshIoBroker stkIo(communicator);
69      const std::string exodusFileName = "generated:1x1x8";
70      size_t index = stkIo.add_mesh_database(exodusFileName, stk::io::READ_MESH);
71      stkIo.set_active_mesh(index);
72      stkIo.create_input_mesh();
73      stkIo.populate_bulk_data();
74
75      // ============================================================
76      //+ EXAMPLE
77      //+ Write output file with all parameters in params list...
78      size_t idx = stkIo.create_output_mesh(file_name,
79                                            stk::io::WRITE_RESTART);
```

```
80
81      stk::util::ParameterMapType::const_iterator i  = params.begin();
82      stk::util::ParameterMapType::const_iterator ie = params.end();
83      for (; i != ie; ++i) {
84        const std::string parameterName = (*i).first;
85        stk::util::Parameter &param = params.get_param(parameterName);
86        stkIo.add_global(idx, parameterName, param.value, param.type);
87      }
88
89      stkIo.begin_output_step(idx, 0.0);
90
91      for (i = params.begin(); i != ie; ++i) {
92        const std::string parameterName = (*i).first;
93        stk::util::Parameter &param = params.get_param(parameterName);
94        stkIo.write_global(idx, parameterName, param.value, param.type);
95      }
96
97      stkIo.end_output_step(idx);
98    }
99
100   {
101     // ============================================================
102     //+ EXAMPLE
103     //+ Read parameters from file...
104     stk::io::StkMeshIoBroker stkIo(communicator);
105     stkIo.add_mesh_database(file_name, stk::io::READ_MESH);
106     stkIo.create_input_mesh();
107     stkIo.populate_bulk_data();
108
109     stkIo.read_defined_input_fields(0.0);
110
111     stk::util::ParameterMapType::const_iterator i = params.begin();
112     stk::util::ParameterMapType::const_iterator ie = params.end();
113     for (; i != ie; ++i) {
114       const std::string parameterName = (*i).first;
115       stk::util::Parameter &param = params.get_param(parameterName);
116       stkIo.get_global(parameterName, param.value, param.type);
117     }
118
119     // ============================================================
120     //+ VALIDATION
121     stk::util::ParameterList gold_params; // To compare values read
122     gold_params.set_param("PI", 3.14159);  // Double
123     gold_params.set_param("Answer", 42);   // Integer
124     gold_params.set_param("doubles", my_vector);
125     gold_params.set_param("Ages", ages);   // Vector of integers...
126
127     size_t param_count = 0;
128     for (i = params.begin(); i != ie; ++i) {
129       param_count++;
130       const std::string parameterName = (*i).first;
131       stk::util::Parameter &param = params.get_param(parameterName);
132       stk::util::Parameter &gold_parameter =
133         gold_params.get_param(parameterName);
134       validate_parameters_equal_value(param, gold_parameter);
135     }
136
137     std::vector<std::string> globalNamesOnFile;
138     stkIo.get_global_variable_names(globalNamesOnFile);
139     ASSERT_EQ(param_count, globalNamesOnFile.size());
140   }
141   // ============================================================
142   // CLEAN UP
143   unlink(file_name.c_str());
144 }
145
```

4.1.30 Writing global variables automatically

This example is similar to the previous one except that in this case, the global variables are written automatically without calling `write_global()` for each value. The only changes to the previous example are:

- replace the call to `add_global()` with a call to `add_global_ref()`.
- pass the address of the value instead of just the value as is shown on line 94, and
- replace the code on lines 89 to 97 of the previous example with the call to `process_output_request()` on line 99.

Listing 4.35: Automatically writing parameters as global variables ../../../code/stk/stk_io/testsForDocumentation/writingAndReadingGlobalParametersAuto.cpp

```
74    // ... Setup is the same as in the previous example
75    // Write output file with all parameters in params list...
76    {
77      stk::io::StkMeshIoBroker stkIo(communicator);
78      const std::string exodusFileName = "generated:1x1x8";
79      size_t input_index = stkIo.add_mesh_database(exodusFileName, stk::io::READ_MESH);
80      stkIo.set_active_mesh(input_index);
81      stkIo.create_input_mesh();
82      stkIo.populate_bulk_data();
83
84      size_t idx = stkIo.create_output_mesh(file_name,
85                                            stk::io::WRITE_RESTART);
86
87      stk::util::ParameterMapType::const_iterator i = params.begin();
88      stk::util::ParameterMapType::const_iterator iend = params.end();
89      for (; i != iend; ++i) {
90        const std::string paramName = (*i).first;
91        //+ NOTE: Need a reference to the parameter.
92        stk::util::Parameter &param = params.get_param(paramName);
93        //+ NOTE: Calling add_global_ref, passing address of value
94        stkIo.add_global_ref(idx, paramName, &param.value, param.type);
95      }
96
97      //+ All writing of the values is handled automatically,
98      //+ do not need to call write_global
99      stkIo.process_output_request(idx, 0.0);
100   }
101   // ... Reading is the same as in previous example
102
```

4.1.31 Heartbeat output

The Heartbeat periodically outputs user-defined data to either a text or binary (exodus) file. The data are typically defined in `stk::util::Parameter` objects, but raw integer, double, or complex values can also be specified. The format of the heartbeat output is customizable and consists of an optional "legend" followed by one or more lines containing the current value of the registered variables at each time step. The data can be scalars, vectors, tensors, or other composite types consisting of integer, real, or complex values.

The currently defined basic formats for heartbeat output are:

`CSV`	Comma-separated values. The output consists of a header line containing the names of each variable being output. The names are separated by commas. Each data line consists of comma-separated values.
`TS_CSV`	Time-stamped comma-separated values. Similar to the `CSV` format except that each line is preceded by a timestamp showing, by default, the time of day that the line was output in 24-hour format.
`TEXT`	Similar to `CSV` except that tab characters are used to separate the fields instead of commas.
`TS_TEXT`	Similar to `TEXT` except that each line is preceded by a timestamp.
`SPYHIS`	A format that can be plotted by the *spyplot* graphics program.
`BINARY`	The data will be output to an exodus file as global variables. This is sometimes referred to as a "history" file.

The format is specified as the second argument to the `add_heartbeat_output()` command as shown on line 90 in the example below where the `TEXT` format is selected.

The following example shows the basic usage of the heartbeat capability. In the initialization section, the parameters and their values are defined. Note that in addition to scalar values, vectors of values are also supported. The values to be output to the heartbeat file are defined in lines 92 to 103. The values are output at line 111. Note that the application does not have to individually output each value; the heartbeat system does this automatically. The application only has to make sure that the correct value is in the `parameter.value` prior to calling `process_heartbeat_output()`.

Listing 4.36: Writing global variables to a Heartbeat file ../../../code/stk/stk_io/testsForDocumentation/usingHeartbeat.cpp

```
60    stk::util::ParameterList params;
61
62    {
63      // ============================================================
64      //+ INITIALIZATION...
65      // Add some params to write and read...
66      params.set_param("PI", -3.14159);  // Double
67      params.set_param("Answer", 42);   // Integer
68
69      std::vector<double> my_vector;
70      my_vector.push_back(2.78);
71      my_vector.push_back(5.30);
72      my_vector.push_back(6.21);
73      params.set_param("some_doubles", my_vector);   // Vector of doubles
74
75      std::vector<int> ages;
76      ages.push_back(55);
77      ages.push_back(49);
78      ages.push_back(21);
79      ages.push_back(19);
80      params.set_param("Ages", ages);   // Vector of integers
81    }
82
83    {
84      // ============================================================
```

```
85        //+ EXAMPLE USAGE...
86        //+ Begin use of stk io heartbeat file...
87        stk::io::StkMeshIoBroker stkIo(communicator);
88
89        //+ Define the heartbeat output to be in TEXT format.
90        size_t hb = stkIo.add_heartbeat_output(file_name, stk::io::TEXT);
91
92        stk::util::ParameterMapType::const_iterator i = params.begin();
93        stk::util::ParameterMapType::const_iterator iend = params.end();
94        for (; i != iend; ++i) {
95          const std::string paramName = (*i).first;
96          //+ NOTE: A reference to the param is needed here.
97          stk::util::Parameter &param = params.get_param(paramName);
98
99          //+ Tell heartbeat which variables to output at each step...
100         //+ NOTE: The address of the value to be output is needed since the
101         //+       value is output in the process_heartbeat_output call.
102         stkIo.add_heartbeat_global(hb,paramName, &param.value, param.type);
103       }
104
105       // Application's "Execution Loop"
106       int timestep_count = 1;
107       double time = 0.0;
108       for (int step=1; step <= timestep_count; step++) {
109         //+ Now output the global variables...
110         //+ NOTE: All registered global values automatically output.
111         stkIo.process_heartbeat_output(hb, step, time);
112       }
113     }
```

If the `stk::io::TEXT` argument to the `add_heartbeat_output()` function is changed to `stk::io::BINARY`, then the code will output a binary "history" file instead of a text-based file. Similarly for the other formats described above.

4.1.31.1 Change output precision

The default precision of the floating point values written by heartbeat to the non-binary formats is five which gives a number of the form "-1.12345e+00". To change the precision, the application defines the "PRECISION" property prior to creating the heartbeat output. The lines below show how this is done and also select the `CSV` format. These lines would replace line 90 in the previous example.

Listing 4.37: Writing global variables to a Heartbeat file in CSV format with extended precision ../../../code/stk/stk_io/testsForDocumentation/usingHeartbeatCSVChangePrecision.cpp

```
92        //+ Output should have 10 digits of precision (1.0123456789e+00)
93        //+             default precision is 5 digits (1.012345e+00)
94        Ioss::PropertyManager hb_props;
95        hb_props.add(Ioss::Property("PRECISION", 10));
96
97        //+ Define the heartbeat output and the format (CSV)
98        size_t hb =
99          stkIo.add_heartbeat_output(file_name, stk::io::CSV, hb_props);
100
```

4.1.31.2 Change field separator

Other customizations of the output are also possible. The example below shows the lines that would be changed in order to use a vertical bar "|" as the field separator in the `TEXT` format.

Listing 4.38: Writing global variables to a Heartbeat file with a user-specified field separator ../../../code/stk/stk_io/testsForDocumentation/usingHeartbeatOverrideSeparator.cpp

```
92      //+ Use vertical bar as field separator
93      Ioss::PropertyManager hb_props;
94      hb_props.add(Ioss::Property("FIELD_SEPARATOR", " | "));
95      size_t hb =
96        stkIo.add_heartbeat_output(file_name, stk::io::TEXT, hb_props);
97
```

4.1.32 Miscellaneous capabilities

This section describes how to perform some functions that are useful, but don't fit into any of the previous sections.

4.1.32.1 Add contents of a file and/or strings to the information records of a database

The first example shows how to embed the contents of a file into the information records of a results or restart output database. This is done on line 96. This is often useful since it then provides some documentation internal to the database itself showing the commands that were given to the application that created the database. The example also shows (see line 100) how to add a string as an additional information record.

In a parallel run in which the file-per-processor output is being used, the information records are only written to the file on processor 0.

Listing 4.39: Adding the contents of a file to the information records of an output database ../../../code/stk/stk_io/testsForDocumentation/addFileContentsToOutputDatabase.cpp

```
63    // ============================================================
64    //+ SETUP
65    std::string input_file = "application_input_file.i";
66    std::string info1("This is the first line of the input file.");
67    std::string info2("This is the second line of the input file. "
68                      "It is longer than 80 characters, so it should be wrapped.");
69    std::string info3("This is the third line of the input file.");
70    std::string info4("This is the fourth and last line of the input file.");
71
72    std::string additional_info_record = "This is an info record added explicitly,"
73                                         " not from the input file.";
74    {
75      std::ofstream my_file(input_file.c_str());
76      my_file << info1 <<"\n" << info2 <<"\n" << info3 <<"\n" << info4 <<"\n";
77    }
78
79    {
```

```
80      // ============================================================
81      //+ EXAMPLE
82      stk::io::StkMeshIoBroker stkIo(communicator);
83      size_t ifh = stkIo.add_mesh_database("9x9x9|shell:xyzXYZ", "generated",
84                                           stk::io::READ_MESH);
85      stkIo.set_active_mesh(ifh);
86      stkIo.create_input_mesh();
87      stkIo.populate_bulk_data();
88
89      // Output...
90      size_t fh = stkIo.create_output_mesh(filename,
91                                           stk::io::WRITE_RESULTS);
92      Ioss::Region *io_reg = stkIo.get_output_io_region(fh).get();
93
94      //+ Add the data from the file "application_input_file.i"
95      //+    as information records on this file.
96      io_reg->property_add(Ioss::Property("input_file_name",input_file));
97
98      //+ Add the data from the "additional_info_record" vector as
99      //+    information records on this file.
100     io_reg->add_information_record(additional_info_record);
101
102     stkIo.write_output_mesh(fh);
103     // ... Verification deleted
104
```

4.1.32.2 Tell database to overwrite steps instead of adding new steps

The next example shows how to tell an output database (typically restart) to only store a single time step and overwrite this time step each time that a new step is added to the database. This is done by setting the cycle count on the database to one as is shown on line 84. The reason an application would want to do this is to minimize the size of a restart file, but still output restart data periodically in case the analysis job crashes for some reason.

For more robustness, an application might have two or more restart databases active and cycle writing to each database in turn. That is, if the application had two restart databases and it was writing every 0.1 seconds, it would write to the first database at times 0.1, 0.3, 0.5, 0.7; and it would write to the second database at times 0.2, 0.4. 0.6, 0.8. In this scenario, a crash during the output of one database would not affect the other database, so there should always be a database containing valid data.

Listing 4.40: Overwriting time steps instead of adding new steps to a database ../../../code/stk/stk_io/testsForDocumentation/singleStepOnRestart.cpp

```
73      // ... Setup deleted
74      // ============================================================
75      // EXAMPLE USAGE...
76      // Create a restart file,
77      size_t fh = stkIo.create_output_mesh(filename,
78                                           stk::io::WRITE_RESTART);
79      stkIo.add_field(fh, field);
80
81      //+ Set the cycle count to 1.  This will result in a maximum
82      //+ of one step on the output database -- when a new step is
83      //+ added, it will overwrite the existing step.
84      stkIo.get_output_io_region(fh)->get_database()->set_cycle_count(1);
85
```

```
86        // Write multiple steps to the restart file.
87        for (size_t step=0; step < 3; step++) {
88          double time = step;
89          stkIo.begin_output_step(fh, time);
90          stkIo.write_defined_output_fields(fh);
91          stkIo.end_output_step(fh);
92        }
93
94        //+ At this point, there should only be a single state on the
95        //+ restart database. The time of this state should be 2.0.
96        // ... Verification deleted
97
```

The cycle count can be set to any value. In general, if the "analysis" step is "AS" and the cycle count is "CYCLE", then the database step is given by "AS mod CYCLE" where "mod" is the remainder when AS is divided by CYCLE.

This page intentionally left blank.

Chapter 5

STK Search

The STK *Search* module provides a geometric proximity box-box search using various methods as documented below in Listings 5.1 - 5.2. Though there has been work in adding the geometry toolkit (GTK) search into STK, that work is incomplete.

5.1 STK Search: usage examples

STK Search takes two lists of bounding volumes and finds intersections between them. It is generally more efficient to have the first list be larger than the second list.

5.1.1 Using Boost R-tree bounding volume search

Listing 5.1: Using the bounding volume search with the Boost R-tree method ../../../code/stk/stk_search/testsForDocumentation/boundingBoxSearch3D.cpp

```
35 #include <stk_search/CoarseSearch.hpp>
36 #include <stk_search/BoundingBox.hpp>
37 #include <gtest/gtest.h>
38
39 namespace
40 {
41 typedef stk::search::Box<double> Box;
42 typedef stk::search::IdentProc<int, int> Id;
43 void assertPairInResults(Id a, Id b, const std::vector<std::pair<Id, Id> > &searchResults);
44 TEST(StkSearchHowTo, useBoostRtreeSearch)
45 {
46     MPI_Comm comm = MPI_COMM_WORLD;
47     int myProcId = stk::parallel_machine_rank(comm);
48     std::vector<std::pair<Box, Id> > firstList, secondList;
49     Box unitBox(Box::point_type(0, 0, 0), Box::point_type(1, 1, 1));
50     Id firstId(0, myProcId);
51     firstList.push_back(std::make_pair(unitBox, firstId));
52     Id secondId(1, myProcId);
53     secondList.push_back(std::make_pair(unitBox, secondId));
54
55     std::vector<std::pair<Id, Id> > searchResults;
56     stk::search::coarse_search(firstList, secondList, stk::search::BOOST_RTREE, comm,
               searchResults);
57
58     int numProc = stk::parallel_machine_size(comm);
59     for(int procId = 0; procId < numProc; procId++)
```

```
60      {
61          assertPairInResults(Id(0, myProcId), Id(1, procId), searchResults);
62          assertPairInResults(Id(0, procId), Id(1, myProcId), searchResults);
63      }
64  }
65  TEST(StkSearchHowTo, useSphereAndPointBoundingVolumes)
66  {
67      MPI_Comm comm = MPI_COMM_WORLD;
68      int myProcId = stk::parallel_machine_rank(comm);
69      std::vector<std::pair<stk::search::Sphere<double>, Id> > firstList;
70      stk::search::Point<double> center(0, 0, 0);
71      const double radius = 0.5;
72      stk::search::Sphere<double> unitSphere(center, radius);
73      Id firstId(0, myProcId);
74      firstList.push_back(std::make_pair(unitSphere, firstId));
75      std::vector<std::pair<stk::search::Point<double>, Id> > secondList;
76      stk::search::Point<double> point(0.1, 0.2, 0.3);
77      Id secondId(1, myProcId);
78      secondList.push_back(std::make_pair(point, secondId));
79
80      std::vector<std::pair<Id, Id> > searchResults;
81      stk::search::coarse_search(firstList, secondList, stk::search::BOOST_RTREE, comm,
                searchResults);
82
83      int numProc = stk::parallel_machine_size(comm);
84      for(int procId = 0; procId < numProc; procId++)
85      {
86          assertPairInResults(Id(0, myProcId), Id(1, procId), searchResults);
87          assertPairInResults(Id(0, procId), Id(1, myProcId), searchResults);
88      }
89  }
90  void assertPairInResults(Id a, Id b, const std::vector<std::pair<Id, Id> > &searchResults)
91  {
92      std::pair<Id, Id> expectedIntersectionPair(a, b);
93      std::vector<std::pair<Id, Id> >::const_iterator resultsIter =
94              std::find(searchResults.begin(), searchResults.end(), expectedIntersectionPair);
95      bool foundExpectedPairInResults = resultsIter != searchResults.end();
96      ASSERT_TRUE(foundExpectedPairInResults);
97  }
98  }
```

5.1.2 Search method options

Listing 5.2 shows the list of possible search methods.

Listing 5.2: Search method options ../../../code/stk/stk_search/stk_search/SearchMethod.hpp

```
    BOOST_RTREE,
    OCTREE,
    KDTREE,                // Coming soon!
    MORTON_LINEARIZED_BVH  // Coming soon!
```

Chapter 6

STK Util

The STK Util module provides many utility capabilities that are used within STK modules and STK-based applications. The categories of utilities include error-handling, exception handling, execution tracing, application argument processing, parallel operations, timing, string operations, etc. These utilities are candidates for future independent STK modules.

6.1 Using the Diagnostic Timers

The following tests show the basic usage of the Diagnostic Timers.

Listing 6.1: Diagnostic Timers ../../../code/stk/stk_doc_tests/stk_util/TimerHowTo.cpp

```
35 #include <gtest/gtest.h>
36 #include <stk_util/diag/PrintTimer.hpp>
37 #include <stk_util/diag/Timer.hpp>
38 #include <comparison/stringAndNumberComparisons.h>
39
40 namespace
41 {
42
43 #if defined(NDEBUG)
44   const double tolerance = 5e-2;
45 #else
46   const double tolerance = 10e-2;
47 #endif
48
49 void doWork()
50 {
51     ::usleep(1e5);
52 }
53
54 TEST(StkDiagTimerHowTo, useTheRootTimer)
55 {
56     stk::diag::TimerSet enabledTimerSet(0);
57     stk::diag::Timer rootTimer = createRootTimer("totalTestRuntime", enabledTimerSet);
58
59     {
60         stk::diag::TimeBlock totalTestRuntime(rootTimer);
61         doWork();
62
63         std::ostringstream outputStream;
64         bool printTimingsOnlySinceLastPrint = false;
65         stk::diag::printTimersTable(outputStream, rootTimer, stk::diag::METRICS_ALL,
               printTimingsOnlySinceLastPrint);
66
```

```
        std::string expectedOutput = "                                                                  \
                 Timer                  Count       CPU Time              Wall Time       \
---------------------------------------- ----- -------------------- -------------------- \
totalTestRuntime                          1        SKIP  SKIP             0.100 SKIP      \
                                                                                          \
Took 0.0001 seconds to generate the table above.                                          \
    ";
        EXPECT_TRUE(unitTestUtils::areStringsEqualWithToleranceForNumbers(expectedOutput,
            outputStream.str(), tolerance));
    }

    stk::diag::deleteRootTimer(rootTimer);
}

TEST(StkDiagTimerHowTo, useChildTimers)
{
    enum {CHILDMASK1 = 1, CHILDMASK2 = 2};
    stk::diag::TimerSet enabledTimerSet(CHILDMASK1 | CHILDMASK2);
    stk::diag::Timer rootTimer = createRootTimer("totalTestRuntime", enabledTimerSet);
    rootTimer.start();

    stk::diag::Timer childTimer1("childTimer1", CHILDMASK1, rootTimer);
    stk::diag::Timer childTimer2("childTimer2", CHILDMASK2, rootTimer);

    {
        stk::diag::TimeBlock timeStuffInThisScope(childTimer1);
        stk::diag::TimeBlock timeStuffInThisScopeAgain(childTimer2);
        doWork();
    }

    std::ostringstream outputStream;
    bool printTimingsOnlySinceLastPrint = false;
    stk::diag::printTimersTable(outputStream, rootTimer, stk::diag::METRICS_ALL,
            printTimingsOnlySinceLastPrint);

    {
        stk::diag::TimeBlock timeStuffInThisScope(childTimer1);
        doWork();
    }

    stk::diag::printTimersTable(outputStream, rootTimer, stk::diag::METRICS_ALL,
            printTimingsOnlySinceLastPrint);

    std::string expectedOutput = "                                                                  \
                 Timer                  Count       CPU Time              Wall Time       \
---------------------------------------- ----- -------------------- -------------------- \
totalTestRuntime                             1        SKIP   SKIP        0.100 SKIP        \
  childTimer1                                1        SKIP   SKIP        0.100 SKIP        \
  childTimer2                                1        SKIP   SKIP        0.100 SKIP        \
                                                                                          \
Took 0.0001 seconds to generate the table above.                                          \
                 Timer                  Count       CPU Time              Wall Time       \
---------------------------------------- ----- -------------------- -------------------- \
totalTestRuntime                             1        SKIP   SKIP        0.200 SKIP        \
  childTimer1                                2        SKIP   SKIP        0.200 SKIP        \
  childTimer2                                1        SKIP   SKIP        0.100 SKIP        \
                                                                                          \
Took 0.0001 seconds to generate the table above.                                          \
            ";
    EXPECT_TRUE(unitTestUtils::areStringsEqualWithToleranceForNumbers(expectedOutput,
            outputStream.str(), tolerance));

    stk::diag::deleteRootTimer(rootTimer);
}

TEST(StkDiagTimerHowTo, disableChildTimers)
{
    enum {CHILDMASK1 = 1, CHILDMASK2 = 2};
```

```
    stk::diag::TimerSet enabledTimerSet(CHILDMASK2);
    stk::diag::Timer rootTimer = createRootTimer("totalTestRuntime", enabledTimerSet);
    rootTimer.start();

    stk::diag::Timer disabledTimer("disabledTimer", CHILDMASK1, rootTimer);
    stk::diag::Timer enabledTimer("enabledTimer", CHILDMASK2, rootTimer);

    {
        stk::diag::TimeBlock timeStuffInThisScope(disabledTimer);
        stk::diag::TimeBlock timeStuffInThisScopeAgain(enabledTimer);
        doWork();
    }

    std::ostringstream outputStream;
    bool printTimingsOnlySinceLastPrint = false;
    stk::diag::printTimersTable(outputStream, rootTimer, stk::diag::METRICS_ALL,
            printTimingsOnlySinceLastPrint);

    {
        stk::diag::TimeBlock timeStuffInThisScope(disabledTimer);
        doWork();
    }

    stk::diag::printTimersTable(outputStream, rootTimer, stk::diag::METRICS_ALL,
            printTimingsOnlySinceLastPrint);

    std::string expectedOutput = "                                                                          \
                Timer                   Count       CPU Time              Wall Time       \
---------------------------------------- ----- --------------------- --------------------- \
totalTestRuntime                            1        SKIP   SKIP        0.100 SKIP        \
  enabledTimer                              1        SKIP   SKIP        0.100 SKIP        \
                                                                                          \
Took 0.0001 seconds to generate the table above.                                          \
                Timer                   Count       CPU Time              Wall Time       \
---------------------------------------- ----- --------------------- --------------------- \
totalTestRuntime                            1        SKIP   SKIP        0.200 SKIP        \
  enabledTimer                              1        SKIP   SKIP        0.100 SKIP        \
                                                                                          \
Took 0.0001 seconds to generate the table above.                                          \
            ";
    EXPECT_TRUE(unitTestUtils::areStringsEqualWithToleranceForNumbers(expectedOutput,
            outputStream.str(), tolerance));

    stk::diag::deleteRootTimer(rootTimer);
}

}
```

Listing 6.2: Diagnostic Timers in Parallel ../../../code/stk/stk_doc_tests/stk_util/TimerHowToParallel.cpp

```
#include <gtest/gtest.h>
#include <stk_util/diag/PrintTimer.hpp>
#include <stk_util/diag/Timer.hpp>
#include <comparison/stringAndNumberComparisons.h>

namespace
{

const double tolerance = 5e-2;

void doWork()
{
    ::usleep(1e5);
}

TEST(StkDiagTimerHowTo, useTimersInParallel)
```

```
{
    MPI_Comm communicator = MPI_COMM_WORLD;
    int numProcs = -1;
    MPI_Comm_size(communicator, &numProcs);
    if(numProcs == 2)
    {
        enum {CHILDMASK1 = 1};
        stk::diag::TimerSet enabledTimerSet(CHILDMASK1);
        stk::diag::Timer rootTimer = createRootTimer("totalTestRuntime", enabledTimerSet);
        rootTimer.start();

        stk::diag::Timer childTimer1("childTimer1", CHILDMASK1, rootTimer);

        {
            stk::diag::TimeBlockSynchronized
             timerStartSynchronizedAcrossProcessors(childTimer1, communicator);
            doWork();
        }

        std::ostringstream outputStream;
        bool printTimingsOnlySinceLastPrint = false;
        stk::diag::printTimersTable(outputStream, rootTimer, stk::diag::METRICS_ALL,
             printTimingsOnlySinceLastPrint, communicator);

        int procId = -1;
        MPI_Comm_rank(communicator, &procId);
        if(procId == 0)
        {
            std::string expectedOutput = "                                                                    \
                                 CPU Time            CPU Time            CPU Time            \
                                Wall Time           Wall Time           Wall Time            \
     Timer      Count      Sum (% of System)   Min (% of System)   Max (% of System)        \
                           Sum (% of System)   Min (% of System)   Max (% of System)        \
SKIP ----- -------------------- -------------------- SKIP SKIP SKIP -------------------- \
totalTestRuntime  2            SKIP  SKIP          SKIP  SKIP          SKIP  SKIP           \
                               0.200 SKIP          0.100 SKIP          0.100 SKIP           \
  childTimer1     2            SKIP  SKIP          SKIP  SKIP          SKIP  SKIP           \
                               0.200 SKIP          0.100 SKIP          0.100 SKIP           \
                                                                                            \
Took SKIP seconds to generate the table above.                                             \
                  ";
            EXPECT_TRUE(unitTestUtils::areStringsEqualWithToleranceForNumbers(expectedOutput,
             outputStream.str(), tolerance));
        }

        stk::diag::deleteRootTimer(rootTimer);
    }
}

}
```

The line at the end that prints the time to generate the table is not that useful for small or medium sized runs, but at large numbers of processors, it can take a non-trivial amount of time to gather the timing data from all processors. Knowing this time can help you understand the overall problem runtime.

6.2 Communicating with other MPI processors

Listing 6.3 shows an example of how to pass a floating point value(double) to all other processors. Note that there currently is a two phase process for doing this. In phase 1, the data that is to be sent is used to *size* the communication buffer which will be sent to that processor. Then at the end of phase 1, the buffer allocation call is made. Then, in phase 2, the same packing of buffers is done again, and in this phase, the communicate call is made. Finally, the buffer for each receive is obtained and unpacked in the order in which it was packed. Here, the assumption is that only one value is received from each processor.

Note, that the call to *allocate_buffers* takes a parameter which is usually $1/4$ of the total number of processors. Inside the *communicate* method, the number of max processors to communicate with is calculated, and if that number is less than a certain threshold, a sparse communication method is chosen, otherwise a dense communication method is chosen. The parameter sent to *allocate_buffers* is that threshold value.

Listing 6.3: Example showing how to communicate with other processors ../../../code/stk/stk_doc_tests/stk_util/CommSparseHowTo.cpp

```
45 TEST(ParallelComm, HowToCommunicateOneValue)
46 {
47     MPI_Comm comm = MPI_COMM_WORLD;
48     stk::CommSparse commSparse(comm);
49 
50     int myProcId = commSparse.parallel_rank();
51     int numProcs = commSparse.parallel_size();
52 
53     double sendSomeNumber = 100-myProcId;
54 
55     for(int phase = 0; phase < 2; ++phase)
56     {
57         for (int proc=0;proc<numProcs;proc++)
58         {
59             if ( proc != myProcId )
60             {
61                 stk::CommBuffer& proc_buff = commSparse.send_buffer(proc);
62                 proc_buff.pack<double>(sendSomeNumber);
63             }
64         }
65         if(phase == 0)
66         {
67             commSparse.allocate_buffers();
68         }
69         else
70         {
71             commSparse.communicate();
72         }
73     }
74 
75 
76     for (int proc=0;proc<numProcs;proc++)
77     {
78         if ( proc != myProcId )
79         {
80             stk::CommBuffer& dataReceived = commSparse.recv_buffer(proc);
81             double val = -1;
82             dataReceived.unpack(val);
83             EXPECT_EQ(100-proc, val);
84         }
85     }
```

```
86 }
```

Listing 6.4 shows how to receive an unknown amount of data from a processor.

Listing 6.4: Example showing how to communicate an arbitrary amount of data with other processors ../../../code/stk/stk_doc_tests/stk_util/CommSparseHowTo.cpp

```
88 TEST(ParallelComm, HowToCommunicateAnArbitraryNumberOfValues)
89 {
90     MPI_Comm comm = MPI_COMM_WORLD;
91     stk::CommSparse commSparse(comm);
92
93     int myProcId = commSparse.parallel_rank();
94     int numProcs = commSparse.parallel_size();
95
96     double sendSomeNumber = 100-myProcId;
97
98     for(int phase = 0; phase < 2; ++phase)
99     {
100         for (int proc=0;proc<numProcs;proc++)
101         {
102             if ( proc != myProcId )
103             {
104                 stk::CommBuffer& proc_buff = commSparse.send_buffer(proc);
105                 for (int i=0;i<myProcId;i++)
106                 {
107                     proc_buff.pack<double>(sendSomeNumber+i);
108                 }
109             }
110         }
111         if(phase == 0)
112         {
113             commSparse.allocate_buffers();
114         }
115         else
116         {
117             commSparse.communicate();
118         }
119     }
120
121
122     for (int
              procFromWhichDataIsReceived=0;procFromWhichDataIsReceived<numProcs;procFromWhichDataIsReceived++)
123     {
124         if ( procFromWhichDataIsReceived != myProcId )
125         {
126             stk::CommBuffer& dataReceived =
               commSparse.recv_buffer(procFromWhichDataIsReceived);
127             int numItemsReceived = 0;
128             while ( dataReceived.remaining() )
129             {
130                 double val = -1;
131                 dataReceived.unpack(val);
132                 EXPECT_EQ(100-procFromWhichDataIsReceived+numItemsReceived, val);
133                 numItemsReceived++;
134             }
135             int goldNumItemsReceived = procFromWhichDataIsReceived;
136             EXPECT_EQ(goldNumItemsReceived, numItemsReceived);
137         }
138     }
139 }
```

6.3 Using the STK Scheduler

The STK Scheduler provides a capability for scheduling an operation, for example output, that will happen at various periods throughout an analysis. The application can create a scheduler and then set the schedule based on time intervals, explicit times, step intervals, and explicit steps. Multiple scheduling intervals can be specified with different scheduling in each interval. The application can then query the scheduler throughout the analysis and determine whether the scheduled activity should be performed at the current analysis time and step.

This section describes two methods of using the STK Scheduler tool: time-based and step-based scheduling. Examples of time-based and step-based scheduling are provided below to show the behavior of the two methods and the combinations thereof. The figures at the end of the section show differences between time-based and step-based scheduling. One main difference is that with time-based scheduling, the `is_it_time()` function will return 'true" the first time it is called per time period, while the step-based scheduling will return "true" only if the step number is equal to a step period.

In addition to time-based and step-based scheduling, the STK Scheduler state can also be modified via operating system signals and explicit application control; examine the source code to see these additional capabilities.

Listing 6.5: Using the scheduler ../../../code/stk/stk_util/testsForDocumentation/usingScheduler.cpp

```
36 #include <gtest/gtest.h>
37 #include <stk_util/environment/Scheduler.hpp>
38
39 namespace
40 {
41 TEST(StkUtilTestForDocumentation, TimeBasedScheduling)
42 {
43     stk::util::Scheduler scheduler;
44
45     const stk::util::Time startTime = 0.0;
46     const stk::util::Time timeInterval = 1.0;
47     scheduler.add_interval(startTime, timeInterval);
48
49     stk::util::Step timeStep = 0;
50     EXPECT_TRUE(scheduler.is_it_time(0.0, timeStep++));
51     EXPECT_FALSE(scheduler.is_it_time(0.5, timeStep++));
52     EXPECT_TRUE(scheduler.is_it_time(1.0, timeStep++));
53 }
54
55 TEST(StkUtilTestForDocumentation, TimeBasedSchedulingWithTerminationTime)
56 {
57     stk::util::Scheduler scheduler;
58
59     const stk::util::Time startTime = 2.0;
60     const stk::util::Time timeInterval = 10.0;
61     scheduler.add_interval(startTime, timeInterval);
62
63     const stk::util::Time terminationTime = 8.2;
64     scheduler.set_termination_time(terminationTime);
65
66     stk::util::Step timeStep = 0;
67     EXPECT_FALSE(scheduler.is_it_time(startTime - 1.0, timeStep++));
68     const stk::util::Time firstTimeAfterStartTime = terminationTime-0.1;
69     EXPECT_TRUE(scheduler.is_it_time(firstTimeAfterStartTime, timeStep++));
70     const stk::util::Time firstAfterTermination = terminationTime+0.1;
```

```
71      EXPECT_TRUE(scheduler.is_it_time(firstAfterTermination, timeStep++));
72      EXPECT_FALSE(scheduler.is_it_time(terminationTime+0.2, timeStep++));
73  }
74
75  TEST(StkUtilTestForDocumentation, StepBasedScheduler)
76  {
77      stk::util::Scheduler scheduler;
78
79      const stk::util::Step startStep = 0;
80      const stk::util::Step stepInterval = 4;
81      scheduler.add_interval(startStep, stepInterval);
82
83      const stk::util::Time dt = 0.1;
84      for (stk::util::Step timeStep=0;timeStep<100;timeStep+=3)
85      {
86          stk::util::Time time = timeStep*dt;
87          bool check = scheduler.is_it_time(time, timeStep);
88          if ( timeStep % stepInterval == 0 )
89          {
90              EXPECT_TRUE(check);
91          }
92          else
93          {
94              EXPECT_FALSE(check);
95          }
96      }
97  }
98
99  TEST(StkUtilTestForDocumentation, TimeBasedSchedulerWithTwoTimeIntervals)
100 {
101     stk::util::Scheduler scheduler;
102     const stk::util::Time startTime1 = 0.0;
103     const stk::util::Time delta1 = 0.1;
104     scheduler.add_interval(startTime1, delta1);
105     const stk::util::Time startTime2 = 0.9;
106     const stk::util::Time delta2 = 0.3;
107     scheduler.add_interval(startTime2, delta2);
108
109     stk::util::Step timeStep = 0;
110     EXPECT_TRUE(scheduler.is_it_time(0.0, timeStep++));
111     EXPECT_FALSE(scheduler.is_it_time(0.07, timeStep++));
112     EXPECT_TRUE(scheduler.is_it_time(0.14, timeStep++));
113     EXPECT_TRUE(scheduler.is_it_time(0.62, timeStep++));
114     EXPECT_TRUE(scheduler.is_it_time(0.6999999, timeStep++));
115     EXPECT_FALSE(scheduler.is_it_time(0.77, timeStep++));
116     EXPECT_TRUE(scheduler.is_it_time(0.9, timeStep++));
117     EXPECT_FALSE(scheduler.is_it_time(0.97, timeStep++));
118     EXPECT_FALSE(scheduler.is_it_time(1.04, timeStep++));
119     EXPECT_FALSE(scheduler.is_it_time(1.11, timeStep++));
120     EXPECT_TRUE(scheduler.is_it_time(1.27, timeStep++));
121 }
122 }
```

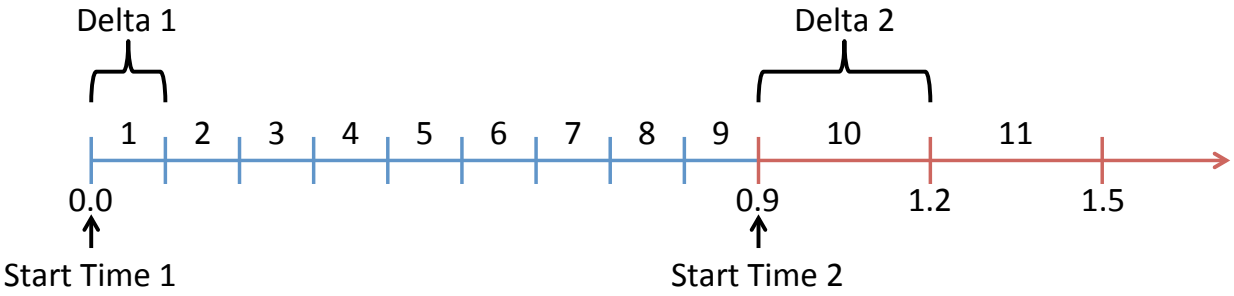

Figure 6.1: Example time-based scheduler: Using two intervals of different sizes. The first interval spans the time from 0.0 to 0.9 with a time-delta of 0.1; the second interval continues from time 0.9 to the end of the analysis with a time-delta of 0.3.

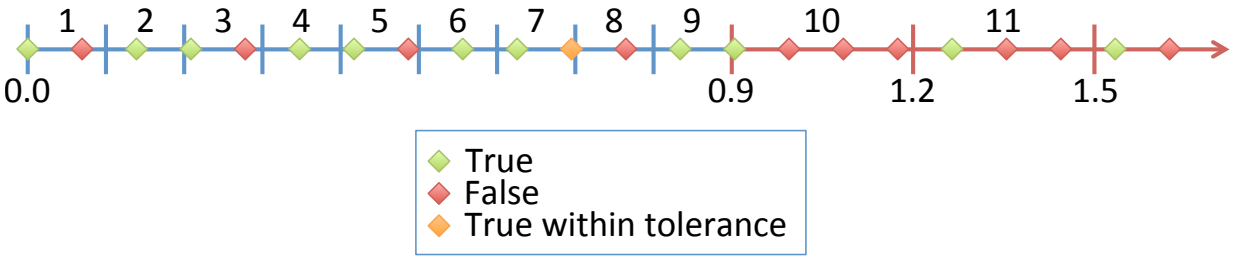

Figure 6.2: Example time-based scheduler: The first call to `is_it_time()` per interval (within a tolerance) will return true. The diamond shapes show the sequence of calls and the color of the diamond signifies whether the function returns true (green or yellow) or false (red). The time-delta and interval settings are the same as in the previous figure.

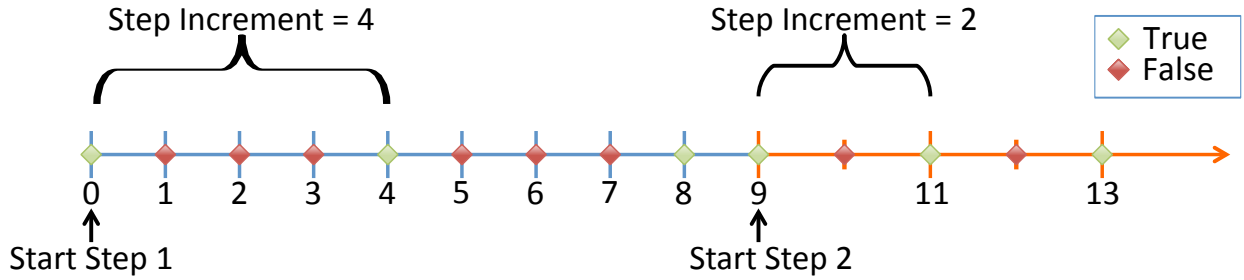

Figure 6.3: Example step-based scheduler: The call to `is_it_time()` will return true on the interval boundary aligned with the step increment. The diamond shapes show the sequence of calls and the color of the diamond signifies whether the function returns true (green) or false (red). This scheduler has two intervals; the first spans steps 0 to 9 with a step-increment of 4 followed by an interval with a step-increment of 2.

6.4 Parameters – type-safe named storage of any variable type

The `Parameter` class provides a type-save mechanism for storing any variable. A variable or vector of variables can be stored in a `ParameterList` and later retrieved by name. The parameters can also be read from and written to mesh and results files as demonstrated in Sections 4.1.29 and 4.1.30.

The supported variable types that can currently be stored in a `Parameter` object are 32-bit integers, 64-bit integers, doubles, floats, and std::strings and vectors of those types. If an additional type is required, it can be added fairly easily and non-supported types can be stored with reduced functionality.

The first example sets up some variables of various types for use in the following parameter examples.

Listing 6.6: Parameters: Data for use in the following examples ../../../code/stk/stk_util/testsForDocumentation/parameters.cpp

```
51      //+ INITIALIZATION
52      std::vector<std::string> expected_name;
53      std::vector<stk::util::ParameterType::Type> expected_type;
54
55      //+ Scalar values of type double, float, int, int64_t, and string
56      double pi = 3.14159;
57      float e = 2.71828;
58      int answer = 42;
59      int64_t big_answer = 42000000000001;
60      std::string team_name = "STK Transition Team";
61
62      expected_name.push_back("PI");
63      expected_type.push_back(stk::util::ParameterType::DOUBLE);
64      expected_name.push_back("E");
65      expected_type.push_back(stk::util::ParameterType::FLOAT);
66      expected_name.push_back("Answer");
67      expected_type.push_back(stk::util::ParameterType::INTEGER);
68      expected_name.push_back("Answer_64");
69      expected_type.push_back(stk::util::ParameterType::INT64);
70      expected_name.push_back("TeamName");
71      expected_type.push_back(stk::util::ParameterType::STRING);
72
73      //+ vector of doubles
74      std::vector<double> my_double_vector;
75      my_double_vector.push_back(2.78); my_double_vector.push_back(5.30);
76      my_double_vector.push_back(6.21);
77      expected_name.push_back("some_doubles");
78      expected_type.push_back(stk::util::ParameterType::DOUBLEVECTOR);
79
80      //+ vector of floats
81      std::vector<float> my_float_vector;
82      my_float_vector.push_back(194.0); my_float_vector.push_back(-194.0);
83      my_float_vector.push_back(47.0);  my_float_vector.push_back(92.0);
84      expected_name.push_back("some_floats");
85      expected_type.push_back(stk::util::ParameterType::FLOATVECTOR);
86
87      //+ vector of ints
88      std::vector<int> ages;
89      ages.push_back(55); ages.push_back(49); ages.push_back(21); ages.push_back(19);
90      expected_name.push_back("Ages");
91      expected_type.push_back(stk::util::ParameterType::INTEGERVECTOR);
92
```

```
93      //+ vector of int64_ts
94      std::vector<int64_t> ages_64;
95      ages_64.push_back(55); ages_64.push_back(49); ages_64.push_back(21);
                ages_64.push_back(19);
96      expected_name.push_back("Ages_64");
97      expected_type.push_back(stk::util::ParameterType::INT64VECTOR);
98
99      //+ vector of strings
100     std::vector<std::string> names;
101     names.push_back("greg"); names.push_back("chloe"); names.push_back("tuffy");
102     names.push_back("liberty"); names.push_back("I have spaces");
103     expected_name.push_back("Names");
104     expected_type.push_back(stk::util::ParameterType::STRINGVECTOR);
105
```

This example illustrates how to create a `ParameterList` and add variables to it. Note that a single `ParameterList` can store multiple variables of multiple types.

Listing 6.7: Parameters: Defining ../../../code/stk/stk_util/testsForDocumentation/parameters.cpp

```
108     //+ Define parameters...
109     stk::util::ParameterList params;
110     params.set_param("PI",           pi);
111     params.set_param("E",            e);
112     params.set_param("Answer",       answer);
113     params.set_param("Answer_64",    big_answer);
114     params.set_param("TeamName",     team_name);
115     params.set_param("some_doubles", my_double_vector);
116     params.set_param("some_floats",  my_float_vector);
117     params.set_param("Ages",         ages);
118     params.set_param("Ages_64",      ages_64);
119     params.set_param("Names",        names);
120
```

Once the parameters have been added to a `ParameterList`, they can be printed or accessed by various means as shown in the following example.

Listing 6.8: Parameters: Accessing values ../../../code/stk/stk_util/testsForDocumentation/parameters.cpp

```
123     //+ Write parameters to stdout...
124     params.write_parameter_list(std::cout);
125
126     //+ Access parameters by name...
127     size_t num_param = expected_name.size();
128     for (size_t i=0; i < num_param; i++) {
129       stk::util::Parameter &param = params.get_param(expected_name[i]);
130       EXPECT_EQ(param.type, expected_type[i]);
131     }
132
133     //+ Extract some parameter values if know type:
134     std::vector<int> pages = params.get_value<std::vector<int> >("Ages");
135     for (size_t i=0; i < pages.size(); i++) {
136       EXPECT_EQ(pages[i], ages[i]);
137     }
138
139     double my_pi = params.get_value<double>("PI");
140     EXPECT_EQ(my_pi, pi);
141
142     //+ Change value of an existing parameter
143     params.set_value("Answer", 21);
144
145     int new_answer = params.get_value<int>("Answer");
146     EXPECT_EQ(new_answer, 21);
```

```
147
148     {
149       //+ Access a variable of unknown type...
150       //+ The parameter uses boost::any to store the actual value.
151       stk::util::Parameter &param = params.get_param("Answer");
152       double value_as_double = 0.0;
153       switch (param.type) {
154       case stk::util::ParameterType::DOUBLE:
155         value_as_double = boost::any_cast<double>(param.value);
156         break;
157       case stk::util::ParameterType::FLOAT:
158         value_as_double = static_cast<double>(boost::any_cast<float>(param.value));
159         break;
160       case stk::util::ParameterType::INTEGER:
161         value_as_double = static_cast<double>(boost::any_cast<int>(param.value));
162         break;
163       case stk::util::ParameterType::INT64:
164         value_as_double = static_cast<double>(boost::any_cast<int64_t>(param.value));
165         break;
166       default:
167         std::cerr << "ERROR: I can not convert 'Answers' value to a double\n";
168         break;
169       }
170       EXPECT_EQ(static_cast<double>(new_answer), value_as_double);
171     }
172
173     {
174       //+ Access a variable of unknown type without using boost::any_cast
175       stk::util::Parameter &param = params.get_param("Answer");
176       double value_as_double = 0.0;
177       switch (param.type) {
178       case stk::util::ParameterType::DOUBLE:
179         value_as_double = params.get_value<double>("Answer");
180         break;
181       case stk::util::ParameterType::FLOAT:
182         value_as_double = static_cast<double>(params.get_value<float>("Answer"));
183         break;
184       case stk::util::ParameterType::INTEGER:
185         value_as_double = static_cast<double>(params.get_value<int>("Answer"));
186         break;
187       case stk::util::ParameterType::INT64:
188         value_as_double = static_cast<double>(params.get_value<int64_t>("Answer"));
189         break;
190       default:
191         std::cerr << "ERROR: I can not convert 'Answers' value to a double\n";
192         break;
193       }
194       EXPECT_EQ(static_cast<double>(new_answer), value_as_double);
195     }
196
197
```

This example shows how the `Parameter` class deals with errors such as accessing nonexistent parameters or specifying the incorrect type for a parameter.

Listing 6.9: Parameters: Dealing with errors
../../../code/stk/stk_util/testsForDocumentation/parameters.cpp

```
200     //+ If the requested parameter does not exist,
201     //+ an error message is printed to stderr and an invalid
202     //+ parameter object is returned
203     stk::util::Parameter no_exist = params.get_param("DoesNotExist");
204     EXPECT_EQ(stk::util::ParameterType::INVALID, no_exist.type);
205
206     //+ In this method of requesting a parameter, no error
```

```
207     //+ message is printed if the parameter doesn't exist and
208     //+ instead the returned iterator is equal to the end of the
209     //+ parameter list.
210     stk::util::ParameterMapType::iterator it = params.find("DoesNotExist");
211     EXPECT_TRUE(it == params.end());
212
213     //+ If the value of a non-existant parameter is requested,
214     //+ an error message is printed and the value 0 is returned.
215     double invalid_value = params.get_value<double>("DoesNotExist");
216     EXPECT_EQ(0.0, invalid_value);
217
218     //+ If the parameter types do not match, an error message is
219     //+ printed and the value 0 of the requested type is returned.
220     int invalid = params.get_value<int>("PI");
221     EXPECT_EQ(0, invalid);
222
223     //+ If the parameter types do not match, an error message is
224     //+ printed and an empty vector of the requested type is returned.
225     std::vector<double> pies = params.get_value<std::vector<double> >("PI");
226     EXPECT_EQ(0u, pies.size());
227
```

Although it is best to use a `ParameterList` with the supported variable types, it can also be used to store types that it does not officially support. The following example shows this capability by storing a value of `std::complex` type. Note that although an unsupported type can be stored and retrieved from a `ParameterList`, it cannot be read from or written to a mesh or results file or printed using the `Parameter` system.

Listing 6.10: Parameters: Storing unsupported types
../../../code/stk/stk_util/testsForDocumentation/parameters.cpp

```
234     //+ Adding a parameter of "unsupported" type...
235     stk::util::ParameterList more_params;
236     std::complex<double> phase(3.14,2.718);
237     more_params.set_param("phase", phase);
238
239     //+ The print system doesn't know about this type, so will print
240     //+ a warning message about unrecognized type.
241     more_params.write_parameter_list(std::cout);
242
243     //+ However, you can still retrieve the value of the parameter
244     //+ if you know what type it is.
245     std::complex<double> my_phase = more_params.get_value<std::complex<double> >("phase");
246     EXPECT_EQ(my_phase, phase);
247
248     //+ The Parameter class won't help you on determining the type,
249     //+ You must know what it is.
250     EXPECT_EQ(more_params.get_param("phase").type, stk::util::ParameterType::INVALID);
251
252     //+ If the wrong type is specified, an exception will be thrown...
253     EXPECT_ANY_THROW(more_params.get_value<std::complex<int> >("phase"));
254
```

6.5 Filename substitution

The filename_substitution function in STK Util provides a basic substitution capability. If the string (typically a filename) passed as an argument to this function contains "special characters",

the special characters will be replaced with runtime-calculated values. The currently supported substitutions are:

- `%B` For applications which use the command-line-argument parsing facilities provided in `stk_util/environment/ProgramOptions.hpp`, and which use a command-line argument called "input-deck", then `%B` will be replaced by the basename of the file named as that "input-deck" argument. If there is no "input-deck" argument, then the basename "stdin" will be used. The basename of the file is the portion of the string between the last "/" and the last ".". For example, given the string `/path/to/the/file/`**`input`**`.i`, the basename would be `input`.
- `%P` will be replaced by the number of processors being used in the current execution.

The example below shows a very simple example of this capability. It is run on 1 processor with no input file, so the substituted filename should be "stdin-1.e".

Listing 6.11: Filename substitution capability
../../../code/stk/stk_util/testsForDocumentation/filenameSubstitution.cpp

```
35 #include <gtest/gtest.h>                // for AssertHelper, EXPECT_EQ, etc
36 #include <stk_util/environment/EnvData.hpp>  // for EnvData
37 #include <stk_util/environment/FileUtils.hpp>
38 #include <stk_util/environment/ProgramOptions.hpp>
39 #include <string>                       // for string, allocator, etc
40 #include <utility>                      // for make_pair
41 #include "boost/program_options/variables_map.hpp"  // for variable_value, etc
42
43 namespace
44 {
45   TEST(StkUtilHowTo, useFilenameSubstitutionWithNoCommandLineOptions)
46   {
47     const std::string default_base_filename = "stdin";
48     const std::string numProcsString = "1";
49     const std::string expected_filename = default_base_filename + "-" + numProcsString + ".e";
50
51     std::string file_name = "%B-%P.e";
52     stk::util::filename_substitution(file_name);
53     EXPECT_EQ(expected_filename, file_name);
54   }
55
56   void setFilenameInCommandLineOptions(const std::string &filename)
57   {
58     boost::program_options::variables_map &command_line_options = stk::get_variables_map();
59     command_line_options.insert(std::make_pair("input-deck",
              boost::program_options::variable_value(filename, false)));
60     stk::EnvData::instance().m_inputFile = filename;
61   }
62   TEST(StkUtilHowTo, useFilenameSubstitutionWithFileComingFromCommandLineOptions)
63   {
64     const std::string base_filename = "myfile";
65     const std::string full_filename = "/path/to/" + base_filename + ".g";
66     setFilenameInCommandLineOptions(full_filename);
67
68     const std::string numProcsString = "1";
69     const std::string expected_filename = base_filename + "-" + numProcsString + ".e";
70
71     std::string file_name = "%B-%P.e";
72     stk::util::filename_substitution(file_name);
73
74     EXPECT_EQ(expected_filename, file_name);
75   }
76 }
```

This page intentionally left blank.

References

[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] Farhat, C. Van der Zee, K.G., Geuzaine, P. "EXODUSII: A Finite Element Data Model Provably second-order time-accurate loosely-coupled solution algorithms for transient nonlinear computational aeroelasticity," Computer Methods in Applied Mechanics and Engineering, 2006; 195 (17-18): 1973-2001

[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.

Index

aura, 20, 23
aura part, 19, 30

buckets, 20
bulkdata, 20

connectivity, 18
custom ghosting, 42

downward relation, 18

element block, 31
entity, 18
explicit member, 32

field, 81
fields, 19

ghosted, 23
ghosting, 20
globally-shared part, 19, 30

induced member, 32, 33

locally-owned part, 19, 30

mesh part, 30
metadata, 20, 36

part, 19
part ordinal, 31
parts, 41
permutations, 18

rank, 18
relations, 18

search, 133
selector, 28
selectors, 19
shared, 23

topology, 18, 63

universal part, 30
upward relation, 18

DISTRIBUTION:

1 MS 0899 Technical Library, 9536 (electronic copy)

This page intentionally left blank.

Sandia National Laboratories