DRAFT: Network Module in GridPACK$^{\mathrm{TM}}$

1 Introduction

This module is designed to create a network, partition it across multiple processors, and allow applications to add and modify fields to the different buses and branches. The core functionality is located in the BaseNetwork class which is a templated class that allows developers to specify arbitrary classes for the buses and branches in the network. Only a single bus or branch object is associated with each bus or branch, so all possible bus and branch behaviors must be incorporated into each bus or branch object. See the section on network components for more information about bus and branch objects. This document will first present information on BaseNetwork methods that are likely to be used by application developers. Additional functionality in the BaseNetwork class that is available but is unlikely to be used outside the GridPACK^{TM} framework itself is presented at the end of this documentation.

The BaseNetwork class is nested within the `gridpack` and `network` namespaces. Within the GridPACK^{TM} framework itself, all BaseNetwork methods are prefixed with `gridpack::network::BaseNetwork` and we encourage developers to use complete namespaces as well. Developers may elect to use the `using` statement instead. Any file creating or using a BaseNetwork object should include the header file

```
#include "gridpack/network/base_network.hpp"
```

As long as the correct location of the `gridpack` link has been specified in the application makefile, all other GridPACK^{TM} include files will automatically be included. The BaseNetwork class can be subclassed to create networks tailored to specific applications, if desired. The BaseNetwork class itself is templated and is declared as

```
template <class _bus, class _branch> class BaseNetwork
```

where `_bus` and `_branch` are component classes representing the behaviors of buses and branches, respectively.

2 Initializing and terminating the network module

A BaseNetwork object can be create and destroyed using simple constructors and destructors. The constructor takes a `parallel::Communicator` object as an argument. This contains the MPI communicator as well as other processor group information and is designed to support the creation of network objects on subsets of the processor world group.

```
explicit void BaseNetwork(
    const gridpack::parallel::Communicator &comm)
comm (IN): communicator on which network is defined
```

This function creates a BaseNetwork object. The BaseNetwork class is a templated class, so actual creation of a base network oject has the form

```
gridpack::network::BaseNetwork<BusClass,BranchClass>
   network(comm);
```

Once created, the network class contains no buses and branches until these are explicitly added. This is usually done by invoking other modules in GridPACK^{TM}, but it can also be done by adding them using functions described below.

```
void ~BaseNetwork(void)
```

The destructor for the BaseNetwork class cleans up all memory used by the BaseNetwork object.

3 Local and global network topology information

These functions provide information about the total number of buses and branches in the entire network as well as the number of buses and branches contained locally on the processor. They also provide information on whether a bus or branch is "owned" by a processor or whether it represents a ghost image of a bus or branch owned by another processor.

`int totalBuses(void)`

This function returns the number of buses in the entire network. This is a global operation so it must be called on all processors in the network. If this number is needed repeatedly, it should be evaluated once and then cached locally.

`int numBuses(void)`

This function returns the number of buses on the calling processor. This includes both local and ghost buses so the sum of the number of buses obtained using the `numBuses` operation across all processors will exceed the number of buses obtained using the `totalBuses` call.

`int totalBranches(void)`

This function returns the number of branches in the entire network. This is a global operation so it must be called on all processors in the network. If this number is needed repeatedly, it should be evaluated once and then cached locally.

`int numBranches(void)`

This function returns the number of branches on the calling processor. This includes both local and ghost branches so the sum of the number of branches obtained using the `numBranches` operation across all processors will exceed the number of branches obtained using the `totalBranches` call.