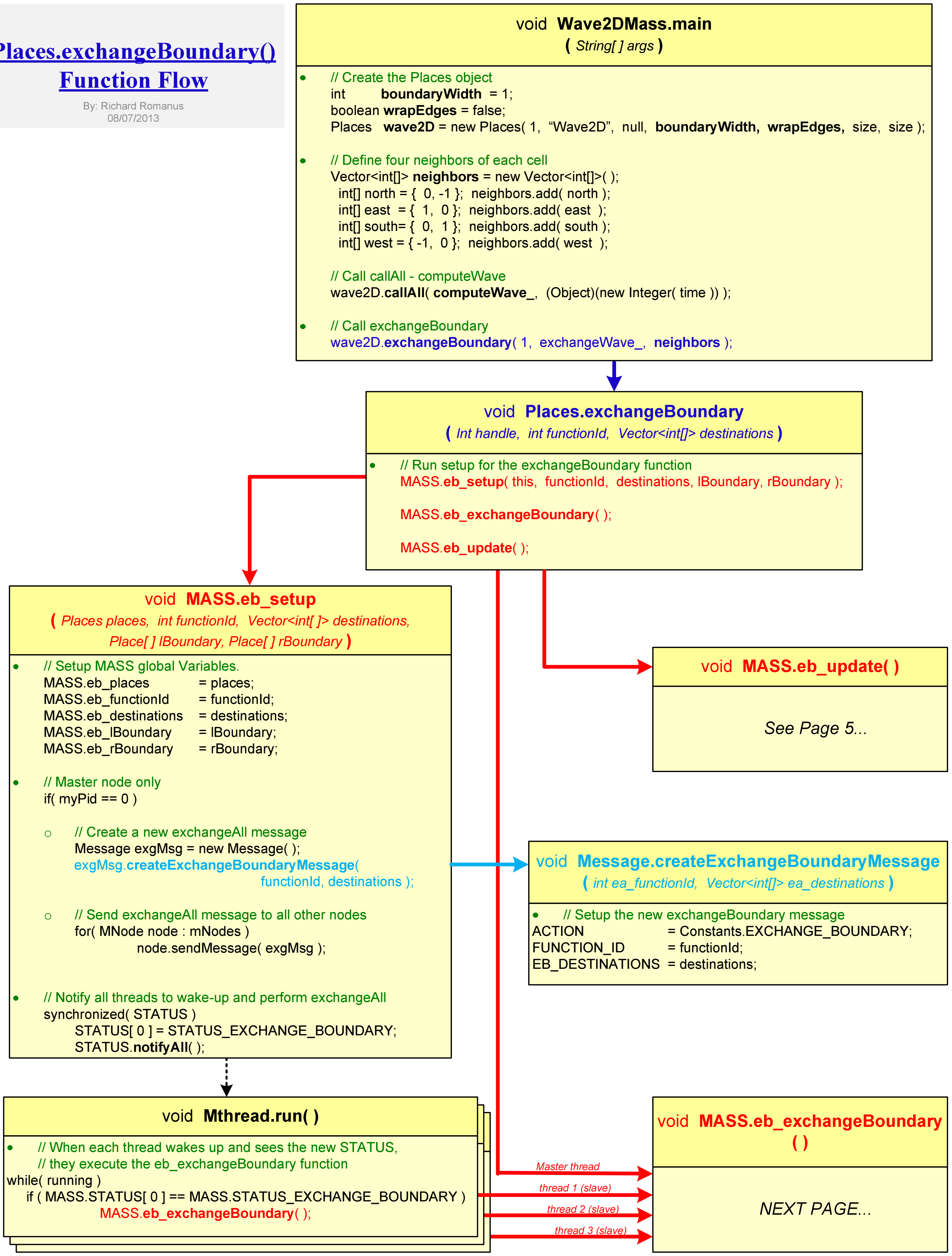
Places.exchangeBoundary() Function Flow

By: Richard Romanus
08/07/2013



```
void  Wave2DMass.main
( String[ ] args )

• // Create the Places object
  int        boundaryWidth  = 1;
  boolean wrapEdges = false;
  Places   wave2D = new Places( 1,  "Wave2D",  null,  boundaryWidth,  wrapEdges,  size,  size );

• // Define four neighbors of each cell
  Vector<int[]> neighbors = new Vector<int[]>( );
    int[] north = {  0, -1 };  neighbors.add( north );
    int[] east  = {  1,  0 };  neighbors.add( east  );
    int[] south= {  0,  1 };  neighbors.add( south );
    int[] west = { -1,  0 };  neighbors.add( west  );

  // Call callAll - computeWave
  wave2D.callAll( computeWave_,  (Object)(new Integer( time )) );

• // Call exchangeBoundary
  wave2D.exchangeBoundary( 1,  exchangeWave_,  neighbors );
```

```
void  Places.exchangeBoundary
( Int handle,  int functionId,  Vector<int[]> destinations )

• // Run setup for the exchangeBoundary function
  MASS.eb_setup( this,  functionId,  destinations, lBoundary, rBoundary );

  MASS.eb_exchangeBoundary( );

  MASS.eb_update( );
```

```
void  MASS.eb_setup
( Places places,  int functionId,   Vector<int[ ]> destinations,
Place[ ] lBoundary, Place[ ] rBoundary )

• // Setup MASS global Variables.
  MASS.eb_places          = places;
  MASS.eb_functionId      = functionId;
  MASS.eb_destinations    = destinations;
  MASS.eb_lBoundary       = lBoundary;
  MASS.eb_rBoundary       = rBoundary;

• // Master node only
  if( myPid == 0 )

    o  // Create a new exchangeAll message
       Message exgMsg = new Message( );
       exgMsg.createExchangeBoundaryMessage(
                                   functionId, destinations );

    o  // Send exchangeAll message to all other nodes
       for( MNode node : mNodes )
               node.sendMessage( exgMsg );

• // Notify all threads to wake-up and perform exchangeAll
  synchronized( STATUS )
       STATUS[ 0 ] = STATUS_EXCHANGE_BOUNDARY;
       STATUS.notifyAll( );
```

```
void  MASS.eb_update( )

See Page 5...
```

```
void  Message.createExchangeBoundaryMessage
( int ea_functionId,  Vector<int[]> ea_destinations )

•  // Setup the new exchangeBoundary message
ACTION                 = Constants.EXCHANGE_BOUNDARY;
FUNCTION_ID            = functionId;
EB_DESTINATIONS  = destinations;
```

```
void  Mthread.run( )

•  // When each thread wakes up and sees the new STATUS,
   // they execute the eb_exchangeBoundary function
while( running )
    if ( MASS.STATUS[ 0 ] == MASS.STATUS_EXCHANGE_BOUNDARY )
                 MASS.eb_exchangeBoundary( );
```

```
void  MASS.eb_exchangeBoundary
( )

NEXT PAGE...
```

void **MASS.eb_exchangeBoundary( )**

```
•   // Some MASS Global Variables
    HashMap< String,  ArrayList<RemoteExchangeRequest> >  exchangeAllRequestMap;

•   // Get thread information
    int numThreads     = threads.size() + 1;
    int threadNumber   = getThreadPosition();

•   // Get Boundary information
    Place shdwPlace;                                                        // the Shadow place
    int lbSize =  (eb_lBoundary == null)  ? 0 : eb_lBoundary.length;        // left Boundary size
    int rbSize =  (eb_rBoundary == null) ? 0 : eb_rBoundary.length;         // right Boundary size
    int totalBndrySize = lbSize + rbSize;                                   // total Boundary size
    int bndryIdx = threadNumber;                                            // set bndryIdx variable to threadnumber

•   // Looping through all of the Place locations managed by this thread
    while( bndryIdx < totalBndrySize )

        // Get the next shadow boundary place
        if( bndryIdx < lbSize )   shdwPlace = eb_lBoundary[ bndryIdx ];            // Left   Boundary
        else                      shdwPlace = eb_rBoundary[ bndryIdx - lbSize ];   // Right Boundary

        // Get shadow global linear index location and the destination hostname
        int shdwGlobalLinearIdx        = Places.getGlobalLinearIndexFromGlobalArrayIndex( shdwPlace.index, eb_places.size() );
        String destHostName            = eb_places.getHostname( shdwGlobalLinearIdx );

        // Create a new request
        RemoteExchangeRequest request = new RemoteExchangeRequest( shdwGlobalLinearIdx,  bndryIdx,  null );
        synchronized(exchangeAllRequestMap)

                // If no requests list exists, create a new one, then add the request to the requests list.
                if( exchangeAllRequestMap.get( destHostName ) == null )
                        ArrayList<RemoteExchangeRequest> requests = new ArrayList<RemoteExchangeRequest>();
                        requests.add( request );
                        exchangeAllRequestMap.put( destHostName, requests );

                // If requests list exists, just add request to it
                else
                        exchangeAllRequestMap.get( destHostName ).add( request );

        // Increment to the next boundary location for this thread
         bndryIdx += numThreads;

•   // Process the Remote Exchange Requests
    barrier( );
    processRemoteExchangeRequest( );
    barrier( );
```

↓

void **MASS.processRemoteExchangeRequest( )**

Not Modified

```
•   // For each of the destination Hosts that has a requestList in the
    // exchangeAllRequestMap, run the startRemoteExchange
    startRemoteExchange( destinationHostName, requestList );
```

↓

void **MASS.startRemoteExchange**
(*String destinationHostName, ArrayList<RemoteExchangeRequest> requestList* **)**

Not Modified

```
•   // Establish connections with destination host
    exchangeHelper[ 0 ].establishConnection( destinationHostName );          →

•   // Create Exchange All request message
    Message  exchangeMsg = new Message( );
    exchangeMsg.createExchangeAllRequestMessage( requestList );

•   // Send request message and process the request
    exchangeHelper[ 0 ].sendRequest( destinationHostName, exchangeMsg );     →
    exchangeHelper[ 0 ].processRequest( destinationHostName );               →
```

void **ExchangeHelper.establishConnection** *Not Modified*
(String hostName)

```
• // Only establish connections to PIDs that are lower than my PID
  if( MASS.nodePidMap.get(hostName) > MASS.myPid )
      return;
```

void **ExchangeHelper.sendRequest** *Not Modified*
(String hostName, Message exgReq)

```
• // Establish connections with destination host
  StreamHandler sh = null;
  synchronized(connectionMap)
  {
      while(connectionMap.get(hostName) == null)
          try { connectionMap.wait(); }
          catch(InterruptedException ie) { }

      sh = connectionMap.get(hostName);
  }

• // Once connection is established, send exchange request message
  sh.sendExchangeRequest(exgReq);
```

void **ExchangeHelper.processRequest** *Not Modified*
(String hostName)

```
• // Establish connections with destination host
  StreamHandler sh = null;
  synchronized( connectionMap )
  {
      while( connectionMap.get( hostName ) == null )
          try { connectionMap.wait(); }
          catch( InterruptedException ie ) { }

      sh = connectionMap.get( hostName );
  }

• // Retrieve exchange requests
  ArrayList<RemoteExchangeRequest> exgReq = sh.readExchangeRequest( );

• // Retrieve the local value
  ArrayList<RemoteExchangeRequest> reqVals = MASS.doRemoteExchangeAll( exgReq );

• // Send local value in a new message to requesting node
  Message exchangeMsg = new Message( );
  exchangeMsg.createExchangeAllRequestMessage( reqVals );
  sh.sendExchangeRequest( exchangeMsg );


• // Read return values from remote request and Update inMessages
  ArrayList<RemoteExchangeRequest> retVals = sh.readExchangeRequest( );
  MASS.updateInMessages( retVals );
```

ArrayList<RemoteExchangeRequest> **MASS.doRemoteExchangeAll**
(*ArrayList<RemoteExchangeRequest> requestList* **)**

```
•   // Establish connections with destination host
    Place destination;
    RemoteExchangeRequest returnReq;
    ArrayList<RemoteExchangeRequest> retList = new ArrayList<RemoteExchangeRequest>();

•   // Loop through each of the request in the request list
    for( RemoteExchangeRequest request : requestList )

        int tmpDestGlbLinIdx = request.getDestinationGlobalLinearIndex();
        int tmpDestLocLinIdx = Places.getLocalLinearIndexFromGlobalLinearIndex( tmpDestGlbLinIdx );

        // If request is a exchangeBoundary request
        if( request.isBoundaryRqst() )
        {
                destination                 = eb_places.get( tmpDestLocLinIdx );

                Object returnMessage        =  destination.outMessages;
                //Object returnMessage      = destination.callMethod( eb_functionId, request.getOutMessage() );

                returnReq                   = new RemoteExchangeRequest(        request.getDestinationGlobalLinearIndex(),
                                                                                request.getOriginGlobalLinearIndex(),
                                                                                returnMessage );
         }
        // If request is an exchangeAll request
        else
        {
                destination                 = ea_places.get( tmpDestLocLinIdx );

                Object returnMessage        =  destination.outMessages;
                //Object returnMessage      = destination.callMethod( ea_functionId, request.getOutMessage() );

                returnReq                   = new RemoteExchangeRequest(        request.getDestinationGlobalLinearIndex(),
                                                                                request.getOriginGlobalLinearIndex(),
                                                                                request.getInMessageIndex(),
                                                                                returnMessage  );
        }
        // Add the return value to the return value list
        retList.add(returnReq);

•   // Return the values
    return retList;
```

void **MASS.updateInMessages**
(*ArrayList<RemoteExchangeRequest> requestList* **)**

```
•   // Establish connections with destination host
    Place origin;
    ArrayList<RemoteExchangeRequest> retList = new ArrayList<RemoteExchangeRequest>();

•   // Loop through each of the return values
    for(RemoteExchangeRequest request : requestList)
    {
        // If request is a exchangeBoundary request
        if( request.isBoundaryRqst() )
        {
                int idx       = request.getBndryIndex();
                int lbSize  = (eb_lBoundary == null) ? 0 : eb_lBoundary.length;

                if( idx < lbSize )      origin = eb_lBoundary[ idx ];
                else                    origin = eb_rBoundary[ idx - lbSize ];

                origin.outMessages = request.getOutMessage();
        }

        // If request is an exchangeAll request
        else
        {
                origin = ea_places.get( Places.getLocalLinearIndexFromGlobalLinearIndex( request.getOriginGlobalLinearIndex() ) );

                origin.inMessages[ request.getInMessageIndex() ] = request.getOutMessage();
        }
```

void **MASS.eb_update( )**

```
•   // Some MASS Global Variables
    HashMap< String,  ArrayList<RemoteExchangeRequest> >  exchangeAllRequestMap;

•   // Create Places Iterator for the range of locations managed by this thread
    Places.Iterator origin_iter = eb_places.iterator( getLocalRange( eb_places ) );

•   // Setup Variables
    int[ ] size = eb_places.size( );                  // size of each dimension
    Place origin;                                     // for caller MASS.Place
    Place dest;                                       // for callee MASS.Place
    int in_msgs_len = eb_destinations.length;         // number of destinations (neighbors)


•   // Looping through all of the Place locations managed by this thread
    while ( origin_iter.hasNext( ) )

    o   // Get the next Place location
        origin = origin_iter.next( );

    o   // Verify the inMessages array and index variable are initialized
        if ( origin.inMessages == null || origin.inMessages.length != in_msgs_len )
                origin.inMessages = new Object[ in_msgs_len ];

        int inMessagesIndex = 0;


    o   // Loop through each of the four destinations (neighbors)
        for( int dest_i = 0;  dest_i < in_msgs_len;  dest_i++ )

        ▪   // Retrieve the destination Coordinates,  Global Linear Index,  and Local Linear Index
            int[ ] neighborCoord                = Places.getGlobalNeighborArrayIndex( origin.index, ea_destinations[ dest_i ], size );   →
            int    globalLinearIndex            = Places.getGlobalLinearIndexFromGlobalArrayIndex( neighborCoord, size );                →
            int    destinationLocalLinearIndex  = Places.getLocalLinearIndexFromGlobalLinearIndex( globalLinearIndex );                  →

        ▪   // On Destination Machine
            if(  ( 0 <= destinationLocalLinearIndex )  &&( destinationLocalLinearIndex < eb_places.length() ) )

                dest = eb_places.get( destinationLocalLinearIndex );
                origin.inMessages[ inMessagesIndex ] = dest.callMethod( eb_functionId, origin.outMessages );

        ▪    // Left Shadow Boundary
            else if( ( destinationLocalLinearIndex < 0 ) && ( eb_lBoundary.length + destinationLocalLinearIndex  < eb_lBoundary.length ) )

                dest = eb_lBoundary[ eb_lBoundary.length + destinationLocalLinearIndex ];
                origin.inMessages[ inMessagesIndex ] = dest.outMessages;

        ▪   // Right Shadow Boundary
            else if( ( destinationLocalLinearIndex >= eb_places.length() ) &&
                                                  ( destinationLocalLinearIndex - eb_places.length() ) < eb_rBoundary.length )

                dest = eb_rBoundary[ destinationLocalLinearIndex - eb_places.length() ];
                origin.inMessages[ inMessagesIndex ] = dest.outMessages;

        ▪   // Error – not found
            else
                origin.inMessages[ inMessagesIndex ] = null;

    o   // Increment the message index
          inMessagesIndex++;
```

class **RemoteExchangeRequest**
implements Serializable

- // Variables

```
private Boolean shadowBoundary;
private int destinationGlobalLinearIndex;
private int originGlobalLinearIndex;
private int inMessageIndex;
private Object outMessage;
```

- // Remote Exchange Request

```
public RemoteExchangeRequest( int destIndex, int origIndex, int inMsgIndex, Object outMsg )
{
        shadowBoundary                 = false;
        destinationGlobalLinearIndex   = destIndex;
        originGlobalLinearIndex        = origIndex;
        inMessageIndex                 = inMsgIndex;
        outMessage                     = outMsg;
}
```

- // Shadow Boundary Remote Exchange Request

```
public RemoteExchangeRequest(  int destIndex, int bndryIndex, Object outMsg )
{
        shadowBoundary                 = true;
        destinationGlobalLinearIndex   = destIndex;
        originGlobalLinearIndex        = bndryIndex;
        inMessageIndex                 = -1;
        outMessage                     = outMsg;
}
```

- // Functions

```
public Boolean   isBoundaryRqst()                      { return shadowBoundary; }
public int       getDestinationGlobalLinearIndex()     { return destinationGlobalLinearIndex; }
public int       getOriginGlobalLinearIndex()          { return originGlobalLinearIndex; }
public int       getBndryIndex()                       { return originGlobalLinearIndex; }
public int       getInMessageIndex()                   { return inMessageIndex; }
public Object    getOutMessage()                       { return outMessage; }
```

int[ ] **Places.getGlobalNeighborArrayIndex**
(int[ ] start, int[ ] offsets, int[ ] size)

Not Modified

```
•   // Create a local destination array for the destination coordinates
    int[ ] destination = new int[ size.length ];

•   // Loop through each of the offsets, calculating the destination
    // If the destination is not valid, throw an exception
    try
          for( int i = 0;  i < offsets.length;  i++ )
                      destination[ i ] = start[ i ] + offsets[ i ];

          if ( destination[ i ] < 0  ||  destination[ i ] >= size[ i ] )
                      throw new Exception( );

•   // If an exception is thrown, the destination array values are all set to -1
    catch ( Exception e )
          for ( int i = 0;  i < destination.length ;  i++ )
                      destination[ i ] = -1;

•   // Return the destination coordinates array
    return destination;
```

int **Places.getGlobalLinearIndexFromGlobalArrayIndex**
(int[ ] index, int[ ] size)

Not Modified

```
•   // Calculate the Global Linear Index, from the Global Array Index
    int retVal = 0;

    for ( int i = 0;  i < index.length ;  i++ ){

          if ( index[ i ] >= 0  &&  size [ i ] > 0  &&  index[ i ] < size[ i ] )
                      retVal  = retVal * size[ i ];
                      retVal  += index[ i ];


•   // Return the Global Linear Index value
    return retVal;
```

int **Places.getLocalLinearIndexFromGlobalLinearIndex**
(int index)

Not Modified

```
•   // Calculate the Local Linear Index from provided Global Linear Index
    return index - offSet;
```

void **Message.createExchangeAllRequestMessage**
(ArrayList<RemoteExchangeRequest> exchangeReqList)

Not Modified

```
•   // Setup the new exchangeAll request message
    message = new HashMap<String, Object>();
    message.put( Constants.EXCHANGE_ALL_MESSAGE, exchangeReqList );
```